

WL-TR-95-3026

SYNTHETIC TERRAIN IMAGERY
FOR HELMET-MOUNTED DISPLAY, VOLUME 2
SOFTWARE DESIGN DOCUMENT



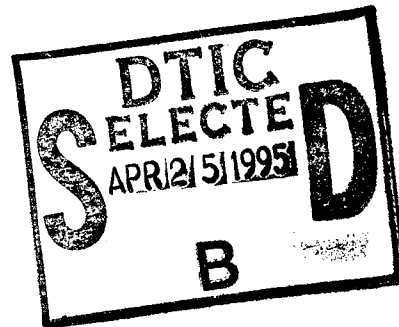
RAND WHILLOCK
BILL CORWIN
JEFF GROAT

HONEYWELL TECHNOLOGY CENTER
3660 TECHNOLOGY DRIVE
MINNEAPOLIS MN 55418

NOVEMBER 1994

FINAL REPORT FOR 06/01/92-03/01/95

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.



DTIC QUALITY INSPECTED 5

FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7562

19950425 096

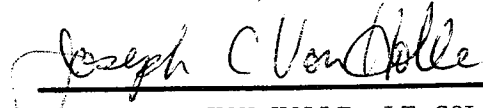
NOTICE

WHEN GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA ARE USED FOR ANY PURPOSE OTHER THAN IN CONNECTION WITH A DEFINITELY GOVERNMENT-RELATED PROCUREMENT, THE UNITED STATES GOVERNMENT INCURS NO RESPONSIBILITY OR ANY OBLIGATION WHATSOEVER. THE FACT THAT THE GOVERNMENT MAY HAVE FORMULATED OR IN ANY WAY SUPPLIED THE SAID DRAWINGS, SPECIFICATIONS, OR OTHER DATA, IS NOT TO BE REGARDED BY IMPLICATION, OR OTHERWISE IN ANY MANNER CONSTRUED, AS LICENSING THE HOLDER, OR ANY OTHER PERSON OR CORPORATION; OR AS CONVEYING ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY IN ANY WAY BE RELATED THERETO.


THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.



ANDREW PROBERT
STI PROJECT ENGINEER
COCKPIT DEVELOPMENT SECTION



JOSEPH C. VON HOLLE, LT COL, USAF
ADVANCED COCKPITS TTIPT LEADER
FLIGHT DYNAMICS DIRECTORATE



RICHARD W. MOSS, CHIEF
COCKPIT DEVELOPMENT SECTION
ADVANCED COCKPIT BRANCH

IF YOUR ADDRESS HAS CHANGED, IF YOU WISH TO BE REMOVED FROM OUR MAILING LIST, OR IF THE ADDRESSEE IS NO LONGER EMPLOYED BY YOUR ORGANIZATION PLEASE NOTIFY WL/FIGP, WRIGHT-PATTERSON AFB, OH 45433-7511 TO HELP MAINTAIN A CURRENT MAILING LIST.

COPIES OF THIS REPORT SHOULD NOT BE RETURNED UNLESS RETURN IS REQUIRED BY SECURITY CONSIDERATIONS, CONTRACTUAL OBLIGATIONS, OR NOTICE ON A SPECIFIC DOCUMENT.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 15 NOV 95	3. REPORT TYPE AND DATES COVERED Software Design Document (6/92--3/95)	
4. TITLE AND SUBTITLE Synthetic Terrain Imagery for Helmet-Mounted display Volume 2 Software Design Document			5. FUNDING NUMBERS F33615-92-C-3601 PE 62201 PR 2403 WU 04 TA SI	
6. AUTHOR(S) Rand Whillock, Bill Corwin, Jeff Groat				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Honeywell Technology Center 3660 Technology Drive Minneapolis, Minnesota 55418			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Flight Dynamics Directorate Wright Laboratory Air Force Materiel Command Wright Patterson AFB OH 45433-7562			10. SPONSORING/MONITORING AGENCY REPORT NUMBER WL-TR-95-3026	
11. SUPPLEMENTARY NOTES WL-TR-95-3025, VOLUME 1, AND WL-TR-95-3027, VOLUME 3, SOFTWARE USER'S MANUAL ARE UNDER SEPARATE COVER.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The software that has been developed as part of the Synthetic Terrain Imagery (STI) Helmet-Mounted Display program is written in C and is designed to execute on a Silicon Graphics VGX Workstation. This software was developed for purposes of evaluating the utility of synthetically derived representations of the local terrain presented on a helmet-mounted display.				
14. SUBJECT TERMS Synthetic Terrain, Synthetic Vision, Helmet-Mounted Display, Advanced Technology, Digital Terrain Elevation Data			15. NUMBER OF PAGES 106	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Table of Contents

<u>Paragraph</u>		<u>Page</u>
1.0	SCOPE	1
1.1	Identification	1
1.2	System Overview	1
1.2.1	Program Overview	1
1.2.2	Program Flow	1
1.2.3	Interfaces	1
1.2.3.1	Inputs	1
1.2.3.1.1	Networked	1
1.2.3.1.2	Standalone	2
1.2.3.2	Output	2
1.2.4	Rendering Routines	2
1.2.5	Data	6
1.2.6	Multiple Processes	6
1.3	Document Overview	6
2.0	REFERENCED DOCUMENTS	7
3.0	PRELIMINARY DESIGN	7
3.1	CSCI Overview	7
3.1.1	CSCI Architecture	8
3.1.2	Systems States and Modes	9
3.1.3	Processing Time Allocation	10
4.0	DETAILED DESIGN	10
4.1.1.1	Design specifications/constraints	10
4.1.1.2	Design	10
4.1.1.2-A	Input/output data elements	10
4.1.1.2-B	Local data elements	12
4.1.1.2-C	Interrupts and signals	12
4.1.1.2-D	Algorithms	13
4.1.1.2-E	Error detection and recovery	17
4.1.1.2-F	Data conversion	17
4.1.1.2-G	Use of other elements	17
4.1.1.2-H	Logic flow	18
4.1.1.2-I	Data structures	18
Appendix A	Software Routines	A-1
Appendix B	Concept Paper	B-1

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

1.0 SCOPE

1.1 Identification

The software that has been developed as part of the Synthetic Terrain Imagery for Helmet-Mounted Display program is written in C and is designed to execute on a Silicon Graphics VGX Workstation. This software was developed for purposes of evaluating the utility of synthetically derived representations of the local terrain presented on a helmet-mounted display.

1.2 System Overview

The software samples data from a preprocessed database created from the U.S. Defense Mapping Agency's Digital Terrain Elevation Data (DTED) database and renders, in perspective view, the visual scene. An NTSC video signal drives a helmet-mounted display.

1.2.1 Program Overview

The Synthetic Terrain Imagery for Helmet Mounted Display program was written to present digital terrain elevation data to pilots in several different formats on an HMD. The program can be run standalone, in which case it behaves like the "flight" program that is provided by Silicon Graphics, or it can be integrated into a simulation environment which communicates over the ethernet.

1.2.2 Program Flow

See section 3.1.2 for a flow diagram of the program.

1.2.3 Interfaces

1.2.3.1 Inputs

The STI program can be compiled for networked operation, where it gets all inputs from other simulations, or it can be compiled for standalone operation by including the -DSTANDALONE flag on the compile line in the makefile.

1.2.3.1.1 Networked

When compiled normally the STI program accesses terrain data in global memory that has been loaded by another program. The STI program opens the terrain data and sets a pointer to the global data structure in the routine `attach_tdb` which is defined in the file `dma.c` and called by the main program `sti.c`

The position and other data for own-ship is also pulled out of global memory. (Actually the data comes in over the Ethernet and is put into global memory by another program. From the STI point of view the data comes from global memory.) The own-ship data is pulled from global memory and processed by the routine `control.c`. At this point the own-ship location coordinates are converted from degrees to feet.

1.2.3.1.2 Standalone

When the -DSTANDALONE flag is included on the compile line in the STI makefile, the program is compiled to run self contained without any other programs running. In this situation the path to a terrain data file must be passed as a parameter on the command line. Own-ship data is calculated internally and the keyboard user interface is active allowing the user to control the own-ship with the mouse and keyboard.

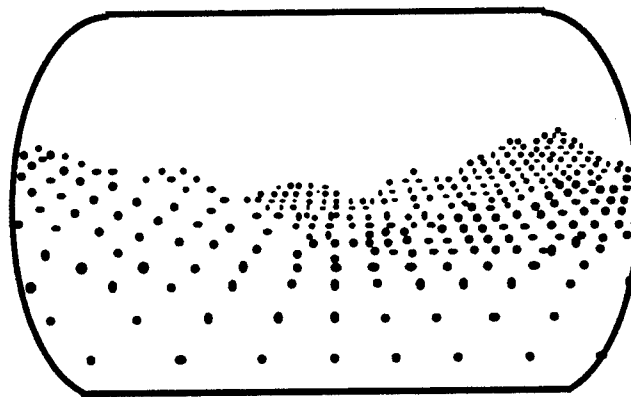
1.2.3.2 Output

The video output of the STI program can take several different forms depending on the command line options specified at start up. If no command line options are given, the video will be full screen 60 HZ high resolution. If the -s (for small window) option is given, the a window the size of an NTSC signal will be opened in the lower left corner of the high resolution monitor. If the -n (ntsc mode) option is given, the video is changed to RS170, and a full sized window will be displayed. This is the video format needed for most HMDs.

1.2.4 Rendering Routines (see section 4.1.1.3-D for detailed algorithms)

The STI program will currently render the terrain in any one of the following formats (subroutines to render the formats in parenthesis):

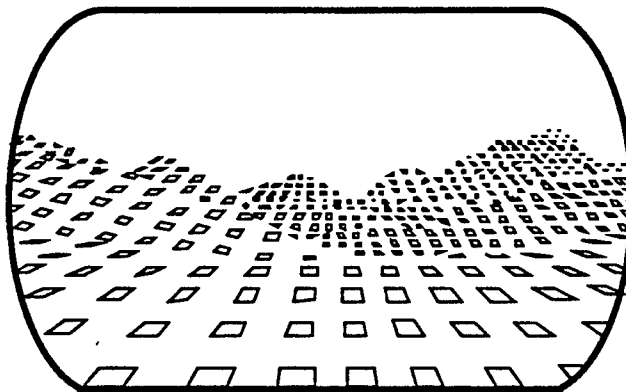
Post tops. In this format the terrain is rendered by drawing a point at the top of each post which is in the view volume. (drawpoints (pg. A-24))



C820574-0

Post tops

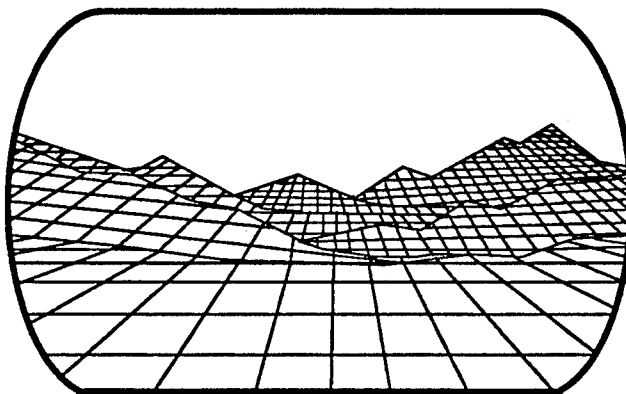
Square post tops (Tiles). This format will draw a square of variable size, as prescribed by the global variable reclen, at the top of each post within the view volume. The elevation data is bilinearly interpolated to obtain the z-value for each corner of the square, so the slope of the square is oriented to the terrain. (drawsquares (pg. A-27))



C920574-0

Tiles

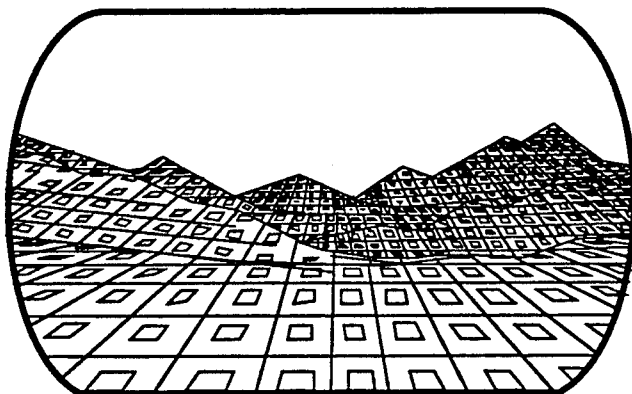
Wireframe mesh. This format is rendered as a set of grids which are fixed to the terrain. It takes two passes through the elevation data to render the mesh. On the first pass, it draws all lines running east-west, and the second pass draws the lines running north-south. (drawmesh (pg. A-22))



C920574-0

Wireframe Mesh

Hybrid mesh and tiles. A wireframe mesh (see above) is first drawn, and then tiles are drawn in the center of the squares formed by the mesh. The tiles are drawn by the same routine which draws the square post tops (see above), but instead of putting the tiles at the post tops, they are offset to fill the gaps formed by the mesh. (drawmesh (pg. A-22) & drawsquares (pg. A-27))



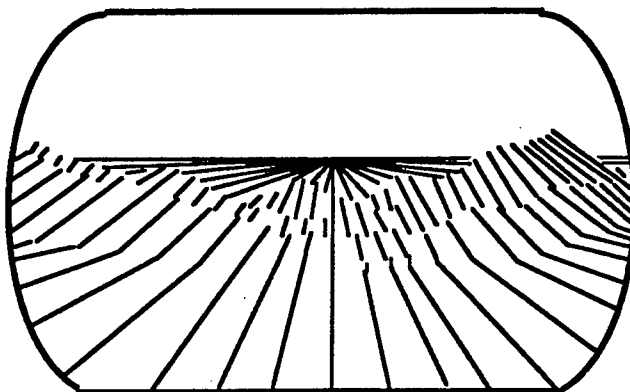
C920574-0

Hybrid Mesh and Tiles

Emergent detail #1. The emergent detail formats are an attempt to provide more detail as the pilot approaches the ground. When the height above ground level (AGL) is greater than 2000 feet, a wireframe mesh is drawn. when agl is between 750 feet and 2000 feet a variant of the hybrid mentioned above is used with points instead of squares in the gaps formed by the mesh. When AGL is below 750 feet, the hybrid format is drawn. (drawmesh (pg. A-22), drawsquares (pg. A-27), and drawpoints (pg. A-24))

Emergent detail #2. Above 2000 feet agl, Post tops are rendered. Between 750 and 2000, the mesh is drawn. Below 750 feet, the hybrid is drawn. (drawmesh (pg. A-22), drawsquares (pg. A-27), and drawpoints (pg. A-24))

Optical expansion gradient. In this format, the terrain is rendered as lines running parallel to the motion of the aircraft. The gradient is fixed to the aircraft, so the elevation data must be bi-linearly interpolated to find points along each line that is to be drawn. (drawgradient (pg. A-18))



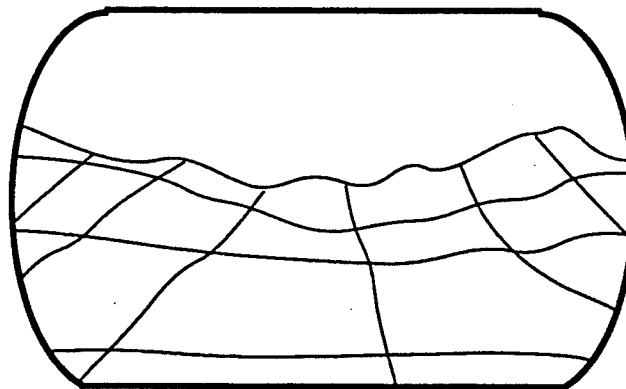
C920574-0

Optical Expansion Gradient

Lighted polygons. The terrain is rendered as lighted shaded polygons using the lighting utilities of the Silicon Graphics' graphics library. The color shading scheme goes from blue in the low regions to green in the middle regions to red in the upper regions (defined in the subroutine initmap (pg. A-40)). This is the only format that uses the surface normals of the elevation posts. A representation of this format is not shown in this document because of the reliance on color for meaningful perception of depth to occur. (drawpolys (pg. A-25))

Shaded polygons. This format displays the terrain as Gouraud shaded polygons. There is no lighting used. See the Description for Lighted polygons for the color scheme. A representation of this format is not shown in this document because of the reliance on color for meaningful perception of depth to occur. (drawpolys (pg. A-25))

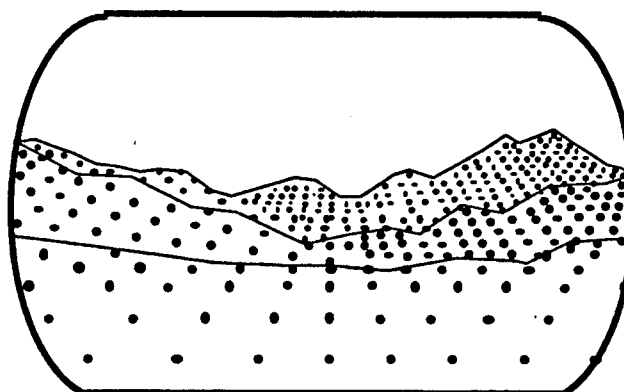
Orthogonal format. In this head referenced format, ridgelines are drawn at 1.5, 3.0, 4.5 and 6.0 NM intervals. Four angular projection lines are then drawn at $\pm 20^\circ$ from the line of sight of the HMD. To facilitate drawing time, one set of point arrays is drawn while a second set is being updated. (Draw_dma_grid (pg. A-16))



BC-HMD-GD-T01

Orthogonal Format

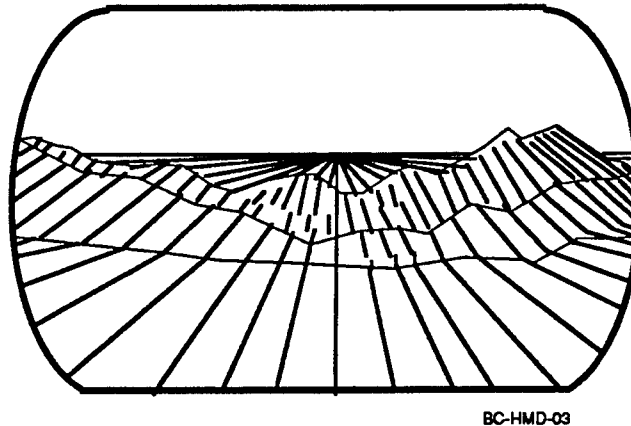
Ridgelines and Post tops. This format renders the terrain as ridgelines plus the post top format to add detail. (drawpoints (pg. A-24) and Draw_dma_ridge (pg. A-17))



BC-HMD-02

Ridgelines with Posttops

Ridgelines and gradient. This format is similar to the orthogonal format. It draws ridgelines with the optical expansion gradient. The difference between this and the orthogonal format is that the spacing may be varied between the gradient lines by changing the global variable skip, while the orthogonal format always has a fixed spacing. (drawgradient (pg. A-18) and Draw_dma_ridge (pg. A-17))



BC-HMD-03

Ridgelines and Gradient

1.2.5 Data

The elevation data used for the simulation is DMA level 1 DTED pre-processed into Lockheed's format. (Lockheed preprocesses the "raw" DMA array into 128-by-128 sample tile to be used in a larger gaming area.) The format is designed to create a regularly structured database, scaled in feet to improve the efficiency of the rendering routines. See section 4.1.1.2-F for a more detailed description of the format.

1.2.6 Multiple Processes

The STI program spawns two processes, both of which are spawned from the main function (defined in STI.c). The first is loadtile, which reads the elevation data from the disk and places it in the appropriate data structure. It communicates with its parent process through the mpflag in the Tile_t structure. The other is gridupdate, which calculates the lines for the orthogonal and ridgeline formats. It also communicates with the parent process through the same flag in the Tile_t structure.

See section 4.1.1.2-I.2 for a description of the Tile_t structure.

1.3 Document Overview

The purpose of this document is to describe the software program (STI).

This document is organized in accordance with Data Information specification DI-MCCR-80012A.

2.0 REFERENCED DOCUMENTS

Appendix A is a complete listing of the headers for the subroutines for the STI software program. The first two pages of Appendix A show the keyboard control commands for the STI program when it is run "stand alone" on a Silicon Graphics workstation.

A "Concept Paper" detailing the technical efforts that are part of the Synthetic Terrain Imagery for Helmet-Mounted Display program aids the reader in understanding this software design document. The Concept Paper is included as Appendix B in this document.

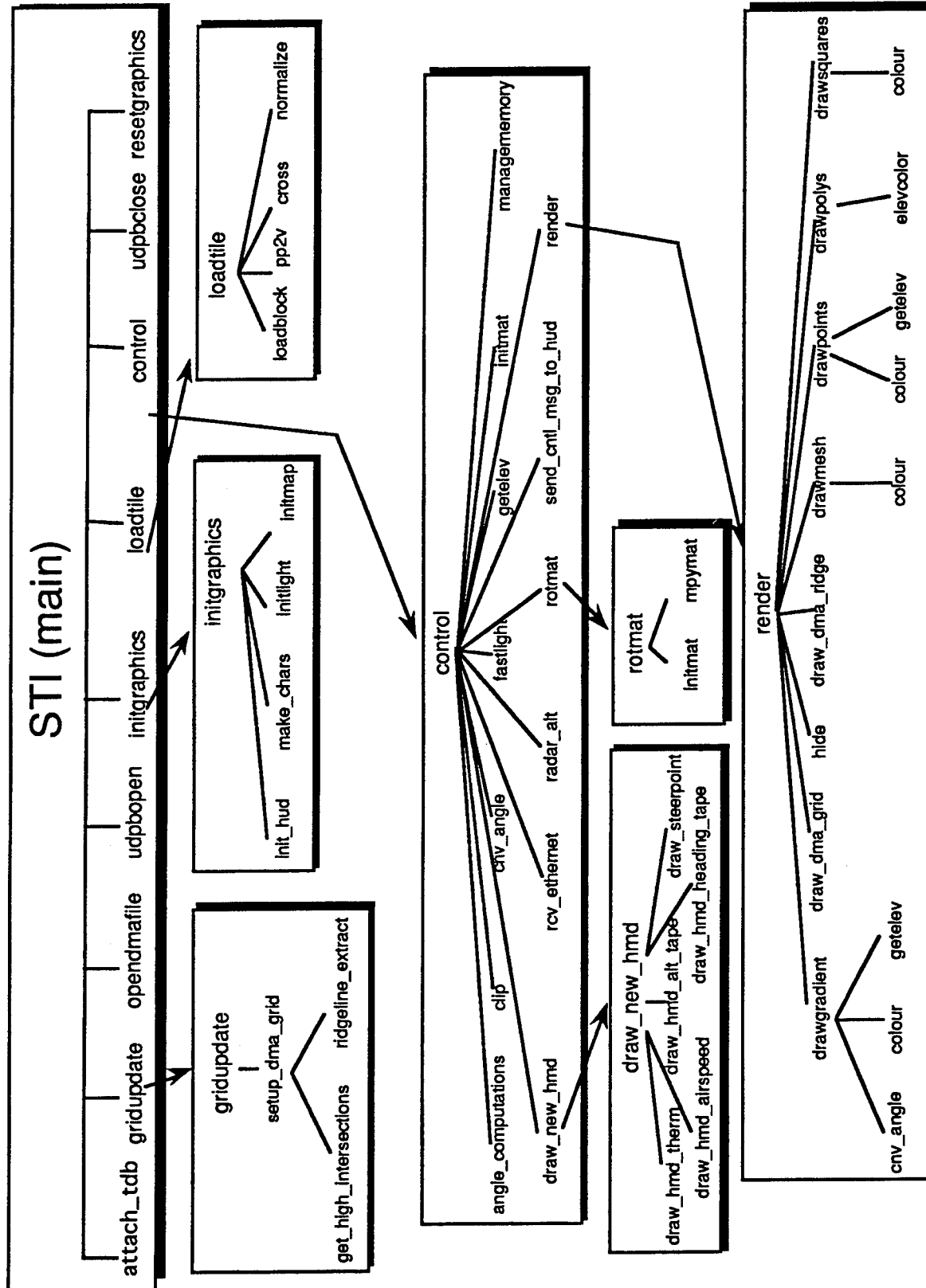
3.0 PRELIMINARY DESIGN

3.1 CSCI Overview

The following sections outline the organizational structure and data flow for the software described in this document.

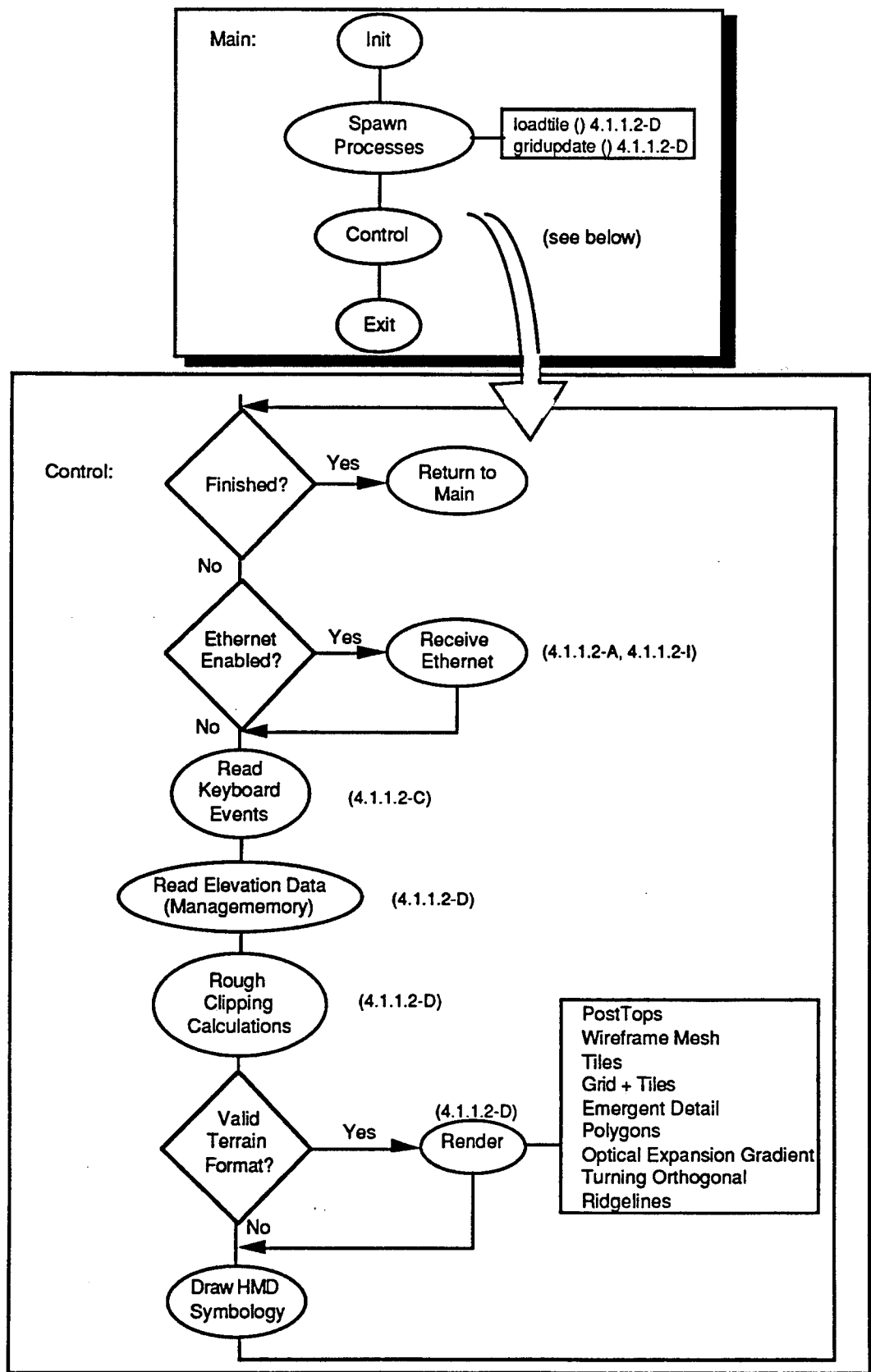
3.1.1 CSCI Architecture

The following chart outlines the organizational structure of the software.

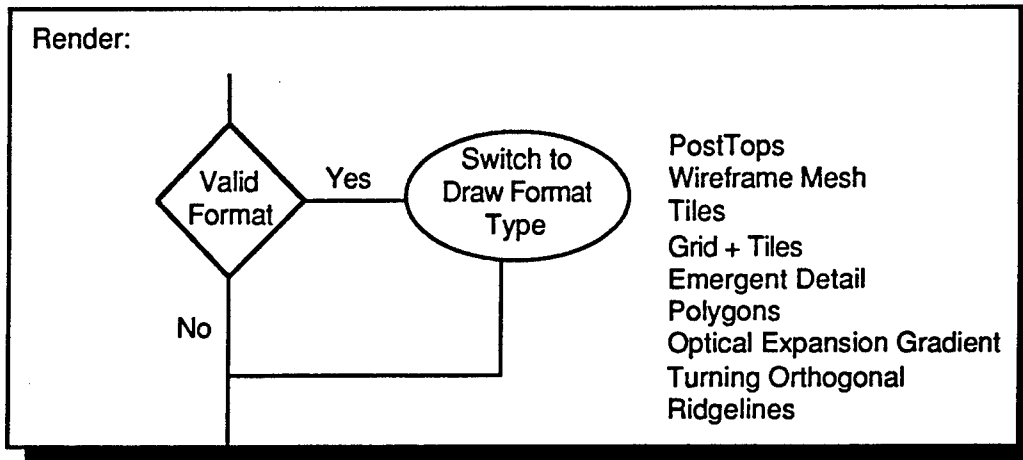


3.1.2 Systems States and Modes

The next illustrations provide an overview of the loop structure for the executable code.



BC421



BC422

3.1.3 Processing Time Allocation

The design goal for the STI program is to update the image at a minimum 30 Hz rate (less than 33 msec frame time). However, to facilitate cases where the available throughput of the Silicon Graphics workstation is unable to render the necessary number of polygons/lines (dependent on the size of the projected view volume onto the terrain and the sample size of the terrain) this rate will decrease. In less demanding conditions, the rate will increase to greater than 30 Hz thereby reducing the latency of the image of the video frame buffer. Nominally, the STI routines will meet this requirement but there are conditions where the effective throughput is insufficient. Optimizing the clipping routines and judicious sampling of the database as controlled by altitude and grazing angle serve to mitigate these effects.

4.0 DETAILED DESIGN

4.1.1.1 Design specifications/constraints

The simulation runs on a Silicon Graphics 4D series computer. There are multiple processes running, so there should be multiple CPUs, otherwise there will be a performance penalty. The memory requirement is 32 Megabytes of memory. Again, the simulation will run with less memory, but there will be a significant performance penalty.

4.1.1.2 Design

The simulation is written in the C programming language. The make utility is used to compile the simulation. The simulation can be compiled for standalone operation by including the `_DSTANDALONE` flag in the makefile.

4.1.1.2-A. Input/output data elements

The data inputs to the STI simulation are specified in the `targets.h` and `terrain.h` include files. This information is copied from global memory during each iteration within `control.c`. Additional data is computed by the routine `Angle_computations` called within `control.c`.

Information specific to the terrain data is described in the include file terrain_db.h written by Lockheed. This information is pulled from global memory and put into local variables by the routine fillglobals called from sti.c.

Global variables

```
float SWLATITUDE;      /* Latitude of SW corner of database
                        (in degrees) */
float SWLONGITUDE;    /* Longitude of SW corner of database
                        (in degrees) */
int  LATRES;          /* Number of points per degree Latitude */
int  LNGRES;          /* Number of points per degree Longitude */
double LAT0;          /* Latitude of SW corner of database
                        (in feet) */
double LNG0;          /* Longitude of SW corner of database
                        (in feet) */
int  xb;              /* beginning x extent of tile to be drawn
                        index into elev. array*/
int  xe;              /* ending x extent of tile to be drawn */
int  yb;              /* beginning y extent of tile to be drawn */
int  ye;              /* ending y extent of tile to be drawn */
int  hidden;          /* hidden point/line removal on/off */
int  skip;            /* distance (in posts) between displayed
                        posts */
int  depth;           /* depth-cueing on/off */
int  fat;             /* lines are fat/skinny */
float position[4];    /* ownship position, x, y, z, x-east, y-north,
                        z-up (in feet)*/
float heading;        /* compass heading of ownship */
float dist;           /* distance we can see (in feet) */
float aspect;         /* aspect ratio (x/y) of the display */
float fov;            /* field of view in the y direction (degrees)*/
float reclen;         /* length of the side of polygonal posts
                        (feet)*/
FILE *dmafp;          /* file pointer to the terrain file */
Tile_t tiles[2];      /* elevation as it is read from disk
                        (ref. 4.1.1.2-I-2)*/
Tile_t *tile;         /* pointer to the current buffer of tiles */

/*The field of view of helmet in x and y for
displaying symbology*/

float hmd_fov_x1, hmd_fov_y1, hmd_fov_x2, hmd_fov_y2;

/* declaration of the point arrays for the orthogonal and ridgeline algs. */
float h[2][5][300][3], vr[2][4][300][3], vl[2][4][300][3];
/* the number of points in each of the above arrays. */
int  np[2][5], npvr[2][4], npvl[2][4];

/* use a double buffered system for the DMA grids. While [toggle]
is being set up, [!toggle] is being drawn. */
int  toggle;
int  format;
```

```

/* index which points to the current buffer being manipulated
by concurrent processes. */
int mpbuf;

struct target_type os;          /* ownship data structure (ref. 4.1.1.2-I.4)*/
FILE *tfile;                   /* output timing file */
long fgmode;                   /* the fog mode currently being used */
float fgparms[5];              /* parameters for the fogvertex call */

```

Formal parameters (see appendix A for parameters to subroutines)

4.1.1.2-B. Local data elements

Local data (see software listing)

4.1.1.2-C. Interrupts and signals

The interrupts and signals to control the simulation are commands received over the ethernet, and operator inputs via the keyboard.

operator key commands:

ESC KEY:	exit
F1 KEY:	set format to point post tops
F2 KEY:	set format to polygonal post tops
F3 KEY:	set format to wireframe mesh
F4 KEY:	set format to wireframe mesh with polygonal posts in the center of the mesh
F5 KEY:	set format to emergent detail, grid > 2000, grid plus points > 750 && < 2000, grid plus square posts < 750
F6 KEY:	set format to emergent detail, points > 2000, grid > 750 && < 2000, grid plus square posts < 750
F7 KEY:	set format to Optical Expansion Gradient
F8 KEY:	set format to Lighted Polygons
F9 KEY:	set format to Elevation shaded Polygons
F10 KEY:	set format to GD's orthogonal format
F11 KEY:	set format to ridgelines + post tops
F12 KEY:	set format to ridgelines + gradient
I/O KEY:	increase/decrease the spacing between posts. spacing can take on the values of 100, 200, 400 or 800 meters.
UP/DOWN ARROW:	increase/decrease the side length of the polygonal posts
C KEY:	toggle depth cueing
Q KEY:	toggle single or double width lines
H KEY:	toggle hidden point/line removal
M/L KEY:	increase/decrease the distance to local horizon
B/D KEY:	brighten/darken the STI
T KEY:	toggle display of terrain on/off
V KEY:	toggle variable local horizon on/off

The following key commands are for STANDALONE operation only:

S/A KEY:	increase/decrease velocity
HOME KEY:	return simulation to original position
L/RT MOUSE:	slew the helmet
MIDDLE MOUSE:	reset helmet to 0
MOUSE POSITION:	
MOUSE-X:	roll of the aircraft
MOUSE-Y:	pitch of the aircraft
	to turn, roll the aircraft about 90 degrees, and then pull back on the mouse.

4.1.1.2-D. Algorithms

Coordinate system. The default coordinate system used in the STI program assumes a flat earth with the x-axis running east-west, x is positive east. The y-axis runs north-south, with north being positive. Z is the elevation and is positive up. All x,y,z coordinates are specified in feet.

The coordinate transform functions used in the STI simulation assume a flat earth model. The functions can be changed to use a universal trans-Mercator model by including the line: #define UTM at the beginning of the routines: loadtile, memory, and control. This model is more accurate at the expense of a noticeable decrease in update rate.

Reading elevation data from disk. The routines loadtile and managememory coordinate to copy elevation data from global memory into local memory when needed. These routine also calculate additional features that will be used when drawing the data. If the UTM flag is defined, the x,y locations for each point are computed using a the universal trans-Mercator projection.

Managememory. The routine managememory (pg. A-46) does the memory management job for the STI program. As input it takes the current position in feet, converts it into file blocks (according to GD's format), and determines if a new tile must be loaded into memory. If a new tile needs to be loaded, it calls the loadtile process indirectly through the mpflag, blkrow, and blkcol fields of the current buffer of the tiles array.

This routine always looks at the same buffer of the tiles array as loadtile, so it can tell loadtile to start reading in a new tile, and also to catch the signal from loadtile that it (loadtile) has completed the load. On the first invocation, managememory must call loadtile and wait until loadtile has finished so the global pointer tile can point to some real data. After the first call, we never wait for loadtile. We simply check to see if it has finished with the current load, and if it has do two things: first make sure tile points to the current buffer (which contains the most recently loaded data), and second check to see if we need to start loading another tile. If another tile is needed, "swap" buffers and give loadtile the appropriate parameters in the tiles array. Let loadtile do its thing while managememory returns control to the calling routine. If loadtile wasn't finished with it's current tile, simply return control to the calling routine.

Loadtile. Loadtile was written to be spawned as a separate process with the `sproc` call. There are no formal parameters, but it communicates with the parent process through several fields of the array tiles.

Loadtile loads the appropriate tile, block by block (calling `loadblock` (pg. A-42)), into memory, sets the appropriate fields of the structure, and computes the surface normals of the terrain at each elevation post.

A "double buffered" approach has been adopted to avoid the problems of multiple processes concurrently accessing and modifying the same memory locations. The parent process will use one buffer while loadtile will modify the other buffer until processing is completed, at which time the buffers are "swapped".

Loadtile will wait until the `mpflag` of the current buffer is set to `MP_CPUSTART`. Once it is told to start, it will get the center row and column in blocks from the `blkrow` and `blkcol` fields. Then it reads data and computes normals. When all processing is done, it sets `mpflag` to `MP_CPUDONE`, signaling the parent process that the tile is loaded and ready for use. It then "swaps" buffers to wait for another `MP_CPUSTART`.

Clipping. The majority of the rendering algorithms use the view volume computed by `clip` (pg. A-6). The view volume is described by the 4 global variables `xb`, `xe`, `yb`, and `ye`. These represent beginning and ending indices into the array of elevation data which is referenced by `tile->elev`.

`Clip` (pg. A-6) performs view volume clipping. For each corner of the screen, calculates the equation of a line from the viewpoint to the corner in 3 space (this line is the intersection of two clipping planes), finds the point on the line that is the far clipping distance away from the viewpoint, and sets (or resets) the `x` and `y` extents of the view volume based on the `x` and `y` coordinates of that point. `X` and `y` of the viewpoint are also included in the `x` and `y` extents to set the near clipping plane.

Sampling. `tile->elev` is a single dimensional pointer to the elevation data to make the storage more dynamic, but represents a two dimensional array. To get the address of a particular post at row `i`, column `j`, an offset is added to `tile->elev` which is equal to $i * \text{the number of east-west posts in the array} + j$. The number of east-west posts in the array is provided by the `ncols` field of the `tile_t` structure. So the address of post $(i,j) = \text{tile->elev} + i * \text{tile->ncols} + j$;

The other global variable accessed by the majority of the rendering algorithms is `skip`. This is the distance between posts which will be drawn. The elevation data being used has a 100 meter resolution, so to calculate the resolution at any `skip` value, simply multiply `skip` by 100 meters. For example, a `skip` of 4 will display posts with a 400 meter resolution. Values that `skip` is allowed to take on are 1, 2, 4, and 8.

Hidden surface removal. Several of the formats need to have hidden points/lines removed. This is accomplished by drawing the terrain as a black surface just below the actual elevation data to allow the `zbuffer` to correctly determine if the point or line should be drawn or not.

Silicon Graphics' graphics library calls referenced.

- cpack - specifies RGBA color with a single packed 32-bit integer
- c3f - sets the RGB values for the current color vector
- n3f - specifies the surface normal of the vertex
- v3f - transfers a 3-D vertex to the graphics pipe
- bgnline, endline - delimit the vertices of a line
- bgnpoint, endpoint - delimit the interpretation of vertex routines as points
- bgntmesh, endtmesh - delimit the vertices of a triangle mesh
- bgnqstrip, endqstrip - delimit the vertices of a quad strip

Post tops. To draw the post tops, first draw the hidden surface, set the color of the terrain, and put the graphics into a point drawing state using the bgnpoint (SG graphics library) call. Loop through all of the posts within the view volume that are "skip" distance apart, and send the vertex of that post down the graphics pipeline using the v3f (SG graphics library) call. When all points to be drawn have been processed, the endpoint command signal is sent (SG graphics library). A variation of this algorithm is used by the emergent detail formats. Instead of turning on the point at the post tops, the points in the middle of the square determined by the four surrounding post tops is turned on. Set a temporary floating point array to the x, y, z values of the point to be drawn, and send that down the graphics pipeline with the v3f (SG graphics library) call.

Square post tops (Tiles). Draw the hidden surface, set up the scalar values to do a bi-linear interpolation of the elevation data, and sets the color of the terrain. For each post that is within the view volume at "skip" distance from its neighbors, define a polygon which has four points (equidistant from the post and aligned to the grid) with a side length of "reclen". Reclen is a global variable that is allowed to be altered by using the up and down arrow keys on the keyboard. Do a bilinear interpolation to get the z-values for the current x, y position. When all four points are defined, put the graphics in polygon mode with the beginpolygon (SG graphics library) call, pass the four vertices down the graphics pipe, and issue the endpolygon (SG graphics library) call to take the graphics out of polygon mode. This format is also used in the emergent detail formats, and the hybrid format, where a square is drawn between the posts instead of at the post tops. To do this, simply add an offset to the x and y values before finding the z-values.

Wireframe mesh. Draw the hidden surface, and set the terrain color. For each line to be drawn in the east-west direction, set the graphics into line drawing mode with the bgnline (SG graphics library) call. Start at the first post within the view volume along this line, and send each post "skip" distance apart down the graphics pipe using the v3f (SG graphics library) call. When all east-west lines have been drawn, repeat the process for the north-south lines.

Hybrid mesh and tiles. See the algorithms for the wireframe mesh and tile formats above.

Emergent detail #1. See the algorithms for the wireframe mesh, post tops, and tile formats above.

Emergent detail #2. See the algorithms for the wireframe mesh, post tops, and tile formats above.

Optical expansion gradient. This format accesses different global variables to be able to draw the scene. `dist` is accessed to determine how long the lines are to be drawn, `heading` is accessed to determine the correct orientation of the lines, and `position` is accessed to compute the starting point for each line. Draw the hidden surface, and set the terrain color. Compute an imaginary horizontal line perpendicular to the current heading that runs through the position point. This line provides the starting x and y coordinates for each line to be drawn. For each point on this line which is skip distance from its neighbor and within the view volume, set the graphics into line drawing mode with the `bgnline` (SG graphics library) call, and step along the line parallel to the heading to get the x, y values of the next point. Call `getelev` (pg. A-31) to do a bilinear interpolation of the data for the z-value. Pass the vertex of the point to the graphics pipeline with the `v3f` (SG graphics library) call. When the line extends beyond the view volume, start on the next point on the imaginary line.

Shaded polygons. Get the first two rows of elevation data within the view volume. Set the graphics in quad strip mode by using issuing the `bgnqstrip` (SG graphics library) command. Step down the rows and draw each polygon defined by the two rows. Set the color at each post based on its elevation (using `elevcolor` (pg. A-28)), and send the vertex down the pipe using the `v3f` (SG graphics library) call. Exit quad strip mode with the `endqstrip` (SG graphics library) command. When the first two rows are finished, use the second and third rows, then the third and fourth, and so on until all rows in the view volume have been drawn.

Lighted polygons. To draw light shaded polygons, a light model has been set up in `light.c`. The algorithm is then the same as drawing shaded polygons with the addition of specifying the surface normals at each point passed down the pipe with the `n3f` (SG graphics library) call. All surface normals have been calculated by `loadtile` (pg. A-43) when the data was read from the disk.

Orthogonal format. The ridgeline extraction system is incorporated into `Setup_dma_grid`, which is called by spawned process `gridupdate`. `Setup_dma_grid` handles both the ridgeline extraction and the turning ortho extraction, the latter by calling `Get_high_intersections`. Both these routines return lines as sets of xyz points to be drawn, using a "doublebuffer" system described more fully below.

Ridgelines and Post tops. See the algorithm for the post top and orthogonal format.

Ridgelines and gradient. See algorithm for the optical expansion gradient and orthogonal format.

Gridupdate. `Gridupdate` is a SPAWNED PROCESS that handles the updates to the GD STI format data extraction systems. It calls `Setup_dma_grid` (pg. A-62) to compute the points for the orthogonal turn and ridgeline formats.

Setup_dma_grid. Depending on whether the STI format is ortho or a ridgeline extraction system, calls `Get_high_intersections` or `Ridgeline_extract` for angles off the current line of sight, extracting the

information needed to draw the lines representing the format and storing them in the h, vr, and vl arrays. When the array is through updating, flips toggle to allow access to the new data.

4.1.1.2-E. Error detection and recovery

In the main (pg. A-44) function, if there are any errors opening the ethernet connection, opening the terrain file, or if the processes could not be spawned, the program will exit gracefully and return to the shell. In the loadtile (pg. A-43) function, the process will exit if a block of memory could not be allocated or read. The getelev (pg. A-31) function checks for several errors. If the Tile_t structures *tile does not point to relevant data, the function returns 0.0, and if the x,y coordinates are not contained in the current tile in memory, out of bounds array indices will be detected, and 0.0 is returned.

4.1.1.2-F. Data conversion

The GD format DMA data is a 32 Mbyte file containing elevations covering about a 3.5 x 3.5 degree area of the world. A utility, Convert_DMA_to_GD_format.c is provided to convert 16 standard DMA elevation files into this format. The files to be converted are listed in "file_list" a separate file. The order is very important, and is described fully in the routine. If you are missing a file needed for an area (not all areas exist), the word "zeroit" may be used to define a 1x1 degree area of elevation zero. (see the Convert_DMA_to_GD_format.c file for further information).

4.1.1.2-G. Use of other elements

Along with the routines detailed in section 4.1.1.2-D, there are a variety of utilities used throughout the software which provide basic functionality. Note that some of the math routines are duplicated. This is due to the fact that the routines have been written to take either floating point or double precision parameters.

Math routines:

- perform vector normalization
 - normalize_vector (double)
 - normalize (float)
- dot product
 - inner_prod
- cross product
 - outer_prod (double)
 - cross (float)
- Matrix operations
 - set_rotation
 - rotate_vector
 - initmat
 - mpymat
- three dimensional distance
 - hypot3
- convert from compass angles to cartesian coordinates
 - cnv_angle

computes a vector between two points
pp2v

Graphics routines:
make_chars
grafstr
centstr

4.1.1.2-H. Logic flow

See processing flow diagram.

4.1.1.2-I. Data structures

The following are data structures defined for the STI program.

4.1.1.2-I.1 Post type is the definition of an elevation data post. It specifies the position of the data post, and its surface normal. The loadtile process reads the elevation data from the disk, and sets the different fields of the data structure based on what was read from the disk. The rendering routines then access the vertices and normals of the data posts as required.

```
typedef struct
{
    float nx;    /* x component of the surface normal (feet)*/
    float ny;    /* y component of the surface normal (feet)*/
    float nz;    /* z component of the surface normal (feet)*/
    float nw;    /* unused */
    float vx;    /* x component of the vertex (feet)*/
    float vy;    /* y component of the vertex (feet)*/
    float vz;    /* z component of the vertex (feet)*/
    float vw;    /* height of the black surface for hidden
                  pt/line removal */
} Post_t;
```

4.1.1.2-I.2 The Tile type structure fully describes the block of elevation data resident in RAM. The STI program uses two of these structures. First, there is an array of size two of these structures. This is to facilitate the "double buffered" approach for multiprocessing. The loadtile process will read data from the disk and set up the structure in one buffer while the rest of the program uses the data in the other buffer. When loadtile is finished loading the current tile, the buffers are swapped, and then loadtile starts loading a new tile while the rest of the program uses the previously loaded tile. The second structure of this type is a pointer. It is always set to point to the most recently loaded buffer of the array mentioned above. This is done so all of the routines don't have to have to keep track of which buffer they are supposed to be using. (See descriptions of managememory and loadtile for further information about the "double buffered" multiprocessing approach.)

```

typedef struct
{
    int mpflag;          /* flag for communication between
                           processes */
    int nrows;           /* number of rows of data posts in
                           this tile */
    int ncols;           /* number of columns of data posts
                           in this tile */
    int blkrow;          /* row number of the center block
                           from the data file */
    int blkcol;          /* column number of the center
                           block from data file */
    float wedge;         /* western edge (in feet) of the tile */
    float sedge;         /* southern edge (in feet) of the tile */
    float minelev;       /* minimum elevation (in feet) of this tile */
    float maxelev;       /* maximum elevation (in feet) of
                           this tile */
    Post_t *elev;        /* pointer to the array of elevation posts */
} Tile_t;

```

4.1.1.2-I.3 The following structure is one of the two structures which are passed over the ethernet. This allows the simulation to get ownship flight parameters. Note that this structure contains a subset of the target_type described below. Also note that the HMD rotation data is provided over the ethernet by the simulation driving the STI program.

The air_pos data is specified in feet, and is relative to an arbitrary origin. The wedge and sedge fields of the terrain_type packet (see below) are offsets specified in the same coordinate system.

```

struct etherown_type
{
    /*****
    *                               *
    *           Flight model structure           *
    *                               *
    *****/

    int    id;
    int    flags;
    int    index; /* target index */
    int    type;  /* target type */

    double    air_pos[3]; /* x, y, z position (feet) */
    double    u0[3];      /* unit vector defining right wing */
    double    u1[3];      /* unit vector defining nose */
    double    u2[3];      /* unit vector normal to airframe. */
    double    velocity[3];

    double    filt_throttle;
    double    fuel;
    double    fuel_flow;
    double    g_norm;
    double    mach;
    double    lookx, looky, lookz; /* HMD rotations (degrees) */
};

```

4.1.1.2-I.4 The target_type structure more fully describes the flight parameters of ownship. The values which correspond to the same fields of the etherown_type are extracted from the etherown_type packet which is sent over the ethernet. The remaining fields can be calculated from the data which was provided over the ethernet.

```

struct target_type
{
/*****
*                               *
***** Flight model structure *****/

    int    id;
    int    flags;
    int    index; /* target index */
    int    type;  /* target type */

    double    air_pos[3];
    double    u0[3];
    double    u1[3];
    double    u2[3];
    double    velocity[3];

    double    filt_throttle;
    double    fuel;
    double    fuel_flow;
    double    g_norm;
    double    mach;
    double    lookx, looky, lookz; /* HMD rotations. */

/* The following variables are USEFUL but can be DERIVED from the
above data; they are NOT sent over ethernet... */

    double    compass;
    double    ground_course;

    double    ground_speed;

    double    airspeed;
    double    roll_angle;
    double    dive_angle;
    double    frame_dive;
    double    side_slip;
    double    aoa;
    double    radar_alt;

    double    stpt_x, stpt_y; /* steerpoint symbol
                               location in HUD
                               coordinates. */

    int alt, vel; /* flags for altimeter option, (RADAR, BARO,
                  AUTO) and as/ option (CAL, TAC, GROUND). */

};

```


4.1.1.2-I.5 And finally, the terrain_type packet is used to describe the preprocessed DTED file. The only parts of the terrain_type packet that are currently used are the wedge and sedge values.

```
/* This is the terrain parameter structure used for terrparm. */
struct terrain_type
{
    int id;
    int flags;

    /* the terrain flags... */
    int wedge, sedge, terrain;
    char terrain_path[80]; /* pathname to the current terrain file.
w/out extension */
    int synterr; /* synthetic terrain format. */
    int hidden; /* hidden surface removal. on/off */
    int depth; /* depthcuing on/off */
    int skip; /* sampling resolution of terrain. */
    int reclen; /* side length of the square posts (feet). */
    int fat; /* toggle double width on/off. */
    int faredge; /* distance top far clipping plane. */
};
```

APPENDIX A

Software Routines

To run:

% STI -ens terrain.dma

Options:

- e enables GD's ethernet protocol. Don't need to supply a file name with this option because a filename is asked for over the ethernet.
- n sets the video output of the computer to RS-170A video format
- s displays the terrain in a RS-170A sized window

Note:

The terrain file is needed in all cases except when using the ethernet.

operator key commands:

ESCKEY:	exit
F1KEY:	set format to point post tops
F2KEY:	set format to polygonal post tops
F3KEY:	set format to wireframe mesh
F4KEY:	set format to wireframe mesh with polygonal posts in the center of the mesh
F5KEY:	set format to emergent detail, grid > 2000, grid plus points > 750 & < 2000, grid plus square posts < 750
F6KEY:	set format to emergent detail, points > 2000, grid > 750 & < 2000, grid plus square posts < 750
F7KEY:	set format to Optical Expansion Gradient
F8KEY:	set format to Lighted Polygons
F9KEY:	set format to Elevation shaded Polygons
F10KEY:	set format to GD's orthogonal format
F11KEY:	set format to ridgelines + post tops
F12KEY:	set format to ridgelines + gradient
I/OKEY:	increase/decrease the spacing between posts. spacing can take on the values of 100, 200, 400 or 800 meters.
UP/DOWNARROW:	increase/decrease the side length of the polygonal posts
CKEY:	toggle depth cueing
PKEY:	toggle horizon line on/off
QKEY:	toggle single or double width lines
HKEY:	toggle hidden point/line removal
M/LKEY:	increase/decrease the distance to local horizon
B/DKEY:	brighten/darken the STI
TKEY:	toggle display of terrain on/off
VKEY:	toggle variable local horizon on/off

S/AKEY:	increase/decrease velocity
HOMEKEY:	return simulation to original position
L/RTMOUSE:	slew the helmet
MIDDLEMOUSE:	reset helmet to 0
MOUSE POSITION:	
MOUSEX:	roll of the plane
MOUSEY:	pitch of the plane
	to turn, roll the plane about 90 degrees, and then pull back on the mouse.

ROUTINE:

Angle_computations

FILE:

ethernet.c

DESCRIPTION:

The data in the packet sent over the ethernet contains the unit vectors determineing the A/C orientation, the velocity vector, and other mathematical data. This routine derives the compass heading, pitch and roll, airspeed, groundspeed, AOA, dive angle, ground course and other "aircraft" data from the raw vector data. BOTH types are used by the system, and to preserve ethernet bandwidth the larger set of data is derived from the smaller.

PARAMETERS:

os wind calc_all_values

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

ROUTINE:

centstr

FILE:

grf.c

DESCRIPTION:

Draws a character string at current location. Leaves the current location at the end of the string. Draws the string from the center of the first letter.

PARAMETERS:

None

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**   chgtercolor
**
** DESCRIPTION:
**   Changes the terrain color from its current value.
**
** PARAMETERS:
**   val      i      relative value to change the terrain color
**
** GLOBALS ACCESSED:
**   tercol    o      packed RGB color
**   intensity i/o    current green intensity of terrain
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/11    original
*/

```

```

/*
** ROUTINE:
**   clip
**
** DESCRIPTION:
**   Performs view volume clipping. For each corner of the screen,
**   calculates the equation of a line from the viewpoint to the
**   corner in 3 space (this line is the intersection of two clipping
**   planes), finds the point on the line that is the local horizon
**   distance away from the viewpoint, and sets (or resets) the x and
**   y extents of the view volume based on the x and y coordinates of
**   that point. x and y of the viewpoint are also included in the x
**   and y extents to set the near clipping plane.
**
** PARAMETERS:
**   pos[3]    i    viewpoint position
**   v0[3]    i    normalized vector pointing to the right
**   v1[3]    i    normalized line of sight vector
**   v2[3]    i    normalized vector pointing up
**   fov      i    field of view in screen y direction
**   aspect   i    aspect ratio of screen x to screen y
**   dist     i    distance to the local horizon
**
** GLOBALS ACCESSED:
**   xb      o    beginning x extent of view volume
**   xe      o    ending x extent of view volume
**   yb      o    beginning y extent of view volume
**   ye      o    ending y extent of view volume
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
**   1.0      dc robohm    92/10  modified so it uses the three unit
**                                vectors, removed procedure calls, and
**                                directly sets the x and y extents.
*/

```

```
/*
** ROUTINE:
**   closedmafile
**
** DESCRIPTION:
**   Closes the file pointed to by dmafp, if dmafp points to a file.
**
** PARAMETERS:
**   None
**
** GLOBALS ACCESSED:
**   dmafp   i       file pointer to open dtd file
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
**/
```



```

/*
** ROUTINE:
**   colour
**
** DESCRIPTION:
**   Sets the current color based on index given.
**
** PARAMETERS:
**   index      i      determines which color to set
**
** GLOBALS ACCESSED:
**   tercol     o      packed RGB color of the terrain
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
**   0.1      dc robohm   92/11  added index for terrain color
*/

```

```

/*
** ROUTINE:
**   control
**
** DESCRIPTION:
**   Polls the keyboard for operator input, updates the position of the
**   plane, updates the tile block in memory, sets up the viewing
**   transformation, clips the terrain to the view volume, and draws
**   the graphics for the current position. This process is continued
**   in a loop until the operator presses the escape key to exit. Once
**   the ethernet code from GD is integrated, the mouse will no longer
**   be used to update the position of the plane.
**
**   operator key commands:
**       ESCKEY:          exit
**       F1KEY:           set format to point post tops
**       F2KEY:           set format to polygonal post tops
**       F3KEY:           set format to wireframe mesh
**       F4KEY:           set format to wireframe mesh with polygonal
**                       posts in the center of the mesh
**       F5KEY:           set format to emergent detail, grid > 2000,
**                       grid plus points > 750 && < 2000, grid plus
**                       square posts < 750
**       F6KEY:           set format to emergent detail, points > 2000,
**                       grid > 750 && < 2000, grid plus square posts < 750
**       F7KEY:           set format to Optical Expansion Gradient
**       F8KEY:           set format to Lighted Polygons
**       F9KEY:           set format to Elevation shaded Polygons
**       F10KEY:          set format to GD's orthogonal format
**       F11KEY:          set format to ridgelines + post tops
**       F12KEY:          set format to ridgelines + gradient
**       I/OKEY:          increase/decrease the spacing between posts.
**                       spacing can take on the values of 100, 200, 400
**                       or 800 meters.
**       UP/DOWNARROW:    increase/decrease the side length of the
**                       polygonal posts
**       CKEY:            toggle depth cueing
**       PKEY:            toggle horizon line on/off
**       QKEY:            toggle single or double width lines
**       HKEY:            toggle hidden point/line removal
**       M/LKEY:          increase/decrease the distance to local horizon
**       B/DKEY:          brighten/darken the STI
**       TKEY:            toggle display of terrain on/off
**       VKEY:            toggle variable local horizon on/off
**
**       S/AKEY:          increase/decrease velocity
**       HOMEKEY:         return simulation to original position
**       L/RTMOUSE:       slew the helmet
**       MIDDLEMOUSE:     reset helmet to 0
**
** PARAMETERS:
**   enet      i      flag to determine ethernet use, or standalone
**                   0 for standalone, !0 for ethernet
**   mode      i      graphics mode, NTSC size or full screen

```

```

**
** GLOBALS ACCESSED:
**   depth      o    depth cueing on or off
**   dist       o    distance to local horizon
**   fat        o    fat lines on or off
**   heading    o    heading of the plane
**   hidden     o    hidden points/lines on or off
**   position   o    position of the plane
**   reclen     o    sidelength of polygonal posts
**   skip       i/o   spacing between posts
**
** USER ROUTINES CALLED:
**   clip
**   chngtercolor
**   cnv_angle
**   fastlight
**   getelev
**   initmat
**   managememory
**   render
**   rotmat
**
** GD ROUTINES CALLED:
**   Angle_computations
**   Draw_new_hmd
**   Radar_alt
**   rcv_ethernet
**   Send_ctrl_msg_to_hud
**
** REVISION HISTORY
**   0.0      dc robohm   92/09   original
**   0.1      dc robohm   92/09   added code to interface ethernet.
**   1.0      dc robohm   92/09   renamed from animate to control
**   1.1      dc robohm   92/11   added for fog for depth cueing
**/

```

```

/*
** ROUTINE:
**   cnv_angle
**
** DESCRIPTION:
**   Converts an angle from a compass heading to cartesian coordinates,
**   and vice-versa and returns the result.
**
** PARAMETERS:
**   angle      i      angle in degrees to be converted
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
*/

```

```

/*
** ROUTINE:
**   cross
**
** DESCRIPTION:
**   Computes the cross product of two vectors.
**
** PARAMETERS:
**   v1      i      vector 1
**   v2      i      vector 2
**   c      o      contains v1 x v2
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
*/

```

```

/*
** ROUTINE:
**   depthcoef
**
** DESCRIPTION:
**   Sets the coefficients for doing linear depth cueing. color values
**   can range from 0..0xff.
**
** PARAMETERS:
**   dist      i      distance over which the colors are linearly mapped
**   minc      i      minimum color. this color will be displayed at
**                     distance dist from viewer
**   maxc      i      maximum color. this color will be displayed at
**                     the viewer position
**
** GLOBALS ACCESSED:
**   c1        o      coefficient. out = in * c1 + c2;
**   c2        o      coefficient. out = in * c1 + c2;
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
*/

```

```

/*
** ROUTINE:
**   depthcolor
**
** DESCRIPTION:
**   Returns the color according to depth. If depth is true, depth
**   cueing is on, so find the distance in the x,y plane (note we are
**   not using 3D distances) and return the color according to
**    $color = dist * c1 + c2$ . Otherwise, return the maximum color,
**   which is defined to be c2. The return value is of the format suited
**   for the cpack call (i.e. 0xAABBGGRR).
**
** PARAMETERS:
**   pnt[3]    i      point for which color needs to be determined
**   depth     i      flag to tell if depth cueing is on or off
**
** GLOBALS ACCESSED:
**   c1        i      coefficient.  $out = dist * c1 + c2$ ;
**   c2        i      coefficient.  $out = dist * c1 + c2$ ;
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09   original
*/

```

```

/*
** ROUTINE:
**   dot
**
** DESCRIPTION:
**   Returns the dot product of two vectors.
**
** PARAMETERS:
**   v1      i      vector 1
**   v2      i      vector 2
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
*/

```


ROUTINE:

Draw_dma_grid

FILE:

GDrender.c

DESCRIPTION:

Takes the arrays as set up by Setup_dma_grid and
Get_high_intersections and draws them.

PARAMETERS:

None

GLOBALS ACCESSED:

toggle

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

Draw_dma_ridge

FILE:

GDrender.c

DESCRIPTION:

Takes the arrays as set up by Setup_dma_grid and
Ridgeline_extract and draws them.

PARAMETERS:

None

GLOBALS ACCESSED:

toggle

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**   drawgradient
**
** DESCRIPTION:
**   Draws the terrain as an optical expansion gradient. Queries the
**   elevation data to get the elevation of points along lines running
**   parallel to the motion of the plane and draws those lines.
**
** PARAMETERS:
**   None
**
** GLOBALS ACCESSED:
**   dist      i      to determine how long the lines are
**   heading   i      to get the orientation of the lines
**   position  i      to get the starting point of each line
**   skip      i      to get the spacing between the lines
**
** USER ROUTINES CALLED:
**   cnv_angle
**   colour
**   getelev
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
**   0.1      dc robohm   92/11  removed call to depthcolor, and added
**                               call to colour since fog is now being
**                               used for depth cueing instead of
**                               computing the color at each vertex
**/

```

ROUTINE:

Draw_hmd_alt_tape

FILE:

hmd.c

DESCRIPTION:

Draws the digital altitude readout.

PARAMETERS:

altitude radar

GLOBALS ACCESSED:

none

USER ROUTINES CALLED:

grafstr

REVISION HISTORY:

0.0 GD original.

NOTES:

Radar is a flag for whether altitude is baro or radar.

ROUTINE:

Draw_hmdmagheading_tape

FILE:

hmd.c

DESCRIPTION:

Draws the heading scale.

PARAMETERS:

planeheading head_adj

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

grafstr

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

Draw_hmd_thermo

FILE:

hmd.c

DESCRIPTION:

Draws the "thermometer" style radar altimeter.

PARAMETERS:

alt

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

grafstr

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**   drawmesh
**
** DESCRIPTION:
**   Draws the terrain as a square grid wireframe mesh.
**
** PARAMETERS:
**   none
**
** GLOBALS ACCESSED:
**   skip      i      determines the resolution of the data samples
**   tile      i      for the elevation data, and array bounds
**   xb        i      beginning x extent of view volume
**   xe        i      ending x extent of view volume
**   yb        i      beginning y extent of view volume
**   ye        i      ending y extent of view volume
**
** USER ROUTINES CALLED:
**   colour
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
**   0.1      dc robohm   92/11  removed call to depthcolor, and added
**                                call to colour since fog is now being
**                                used for depth cueing instead of
**                                computing the color at each vertex
*/

```

ROUTINE:

Draw_new_hmd

FILE:

hmd.c

DESCRIPTION:

Draws the hmd symbology over the STI format.

PARAMETERS:

os, lookvec, hmd_los

GLOBALS ACCESSED:

hmd_fov_x1 hmd_fov_x2 hmd_fov_y1 hmd_fov_y2
offset

USER ROUTINES CALLED:

Draw_hmd_airspeed

Draw_hmd_thermo

Draw_hmd_alt_tape

Draw_hmdmagheading_tape

Draw_steerpoint

REVISION HISTORY:

0.0 GD original.

NOTES:


```

/*
** ROUTINE:
**   drawpoints
**
** DESCRIPTION:
**   Draws the terrain as post tops. Simply turns on one pixel at the
**   desired position. If offset is FALSE, draws the point at the post top,
**   otherwise, draws the point in the center of the four surrounding
**   posts.
**
** PARAMETERS:
**   offset      i      flag to determine if the point are to be drawn
**                       at the posts, or offset between the posts
**
** GLOBALS ACCESSED:
**   skip        i      determines the resolution of the data samples
**   tile         i      for the elevation data, and array bounds
**   xb          i      beginning x extent of view volume
**   xe          i      ending x extent of view volume
**   yb          i      beginning y extent of view volume
**   ye          i      ending y extent of view volume
**
** USER ROUTINES CALLED:
**   colour
**   getelev
**
** REVISION HISTORY
**   0.0         dc robohm   92/09  original
**   0.1         dc robohm   92/11  removed call to depthcolor, and added
**                                   call to colour since fog is now being
**                                   used for depth cueing instead of
**                                   computing the color at each vertex
*/

```

```

/*
** ROUTINE:
**   drawpolys
**
** DESCRIPTION:
**   Draws the terrain as elevation shaded polygons depending on the
**   value of the light flag.
**
** PARAMETERS:
**   light      i      flag to determine light shading
**
** GLOBALS ACCESSED:
**   skip      i      determines the resolution of the data samples
**   tile      i      for the elevation and normal data, and array bounds
**   xb        i      beginning x extent of view volume
**   xe        i      ending x extent of view volume
**   yb        i      beginning y extent of view volume
**   ye        i      ending y extent of view volume
**
** USER ROUTINES CALLED:
**   elevcolor
**
** REVISION HISTORY
**   0.0      dc robohm   92/09   original
*/

```

ROUTINE:

Draw_steerpoint

FILE:

hmd.c

DESCRIPTION:

Draws the steerpoint symbol.

PARAMETERS:

os

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

grafstr

flex_limit_symbol

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**   drawsquares
**
** DESCRIPTION:
**   Draws the terrain as square post tops. If offset is FALSE, draws the
**   square at the post top, otherwise, draws the square in the center
**   of the four surrounding posts.
**
** PARAMETERS:
**   offset      i      flag to determine if the squares are to be drawn
**                       at the posts, or offset between the posts
**
** GLOBALS ACCESSED:
**   skip        i      determines the resolution of the data samples
**   tile         i      for the elevation data, and array bounds
**   xb          i      beginning x extent of view volume
**   xe          i      ending x extent of view volume
**   yb          i      beginning y extent of view volume
**   ye          i      ending y extent of view volume
**
** USER ROUTINES CALLED:
**   colour
**
** REVISION HISTORY
**   0.0          dc robohm    92/09  original
**   1.0          dc robohm    92/10  rewrote the subroutine to remove
**                                   subroutine calls. now do the bilinear
**                                   interpolation within this subroutine so
**                                   we only have to set up the coefficients
**                                   once for every frame, instead of once
**                                   for each of the four corner points of
**                                   each square in the frame.
**   1.1          dc robohm    92/11  removed call to depthcolor, and added
**                                   call to colour since fog is now being
**                                   used for depth cueing instead of
**                                   computing the color at each vertex
*/

```

```

/*
** ROUTINE:
**     elevcolor
**
** DESCRIPTION:
**     Returns the color from sti_colrmap based on the given elevation.
**     the format of the color is suitable for the cpack call (i.e.
**     0xAABBGGRR).
**
** PARAMETERS:
**     z         i         elevation for which the color is needed
**
** GLOBALS ACCESSED:
**     sti_colrmap i         contains the elevation colors
**
** USER ROUTINES CALLED:
**     none
**
** REVISION HISTORY
**     0.0         dc robohm    92/09    original
*/

```

```

/*
** ROUTINE:
**   fastlight
**
** DESCRIPTION:
**   Fastlight is provided so the light may be bound after the
**   transformation matrix has been set up. This is so the light
**   remains fixed to the ground as opposed to being fixed in
**   viewer space.
**
** PARAMETERS:
**   None
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
*/

```

ROUTINE:

flex_limit_symbol

FILE:

hmd.c

DESCRIPTION:

Limits the coordinates passed to it to the passed down field of view; returns true if position was limited.

PARAMETERS:

pos (xy), fovx1, fovx2, fovy1, fovy2

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**     getelev
**
** DESCRIPTION:
**     Does a bi-linear interpolation of the DTED to get and return the
**     elevation at point x,y. If point x,y is not contained in the
**     current tile, 0.0 is returned. Uses the global variable skip to
**     use only the posts that would be present had the data been at
**     skip * 100 resolution.
**
** PARAMETERS:
**     x          i          x position (in feet) of desired elevation
**     y          i          y position (in feet) of desired elevation
**
** GLOBALS ACCESSED:
**     skip       i          determines which posts are used for interpolation
**     tile       i          access wedge, sedge, nrows, and ncols to
**                           determine where this point is in relation to
**                           the current elevation tile, and elev to
**                           determine the elevation.
**
** USER ROUTINES CALLED:
**     None
**
** REVISION HISTORY
**     0.0        dc robohm    92/09    original
*/

```


ROUTINE:

Get_high_intersections

FILE:

GDrender.c

DESCRIPTION:

Get_high_intersections projects from point down los. It stores the point encountered at the highest visual angle so far as it projects. As the projection proceeds, the distance traversed along the HMD los and across it is tracked. As these distances cross the horizontal and vertical criteria passed down by Setup_dma_grid, the current point is added to the appropriate array (h, vl or vr) to extend that line.

PARAMETERS:

point --the starting point of the projection.
los --unit vector defining line of sight to look down.
near --distance criteria for nearest horizontal line.
i1 --distance criteria for 2nd horizontal line.
i2 --for 3rd horizontal line.
far --for 4th horizontal line
s1 --for first vertical line out.
(note: Get_high_intersections is called SEPARATELY to determine the grid to the left and right of the LOS.)
s2 --next vertical line out.
s3, s4 --next vertical line criteria.
vdf --proportion of the projected line in the
--direction of the LOS.
hdf --proportion of the projected line normal to the LOS.
h --array holding the horizontal line points.
v --array holding the vertical line points.
nph --number of points in the horizontal line arrays.
npv --number of points in the vertical line arrays.
flag --determines if left or right of LOS.

GLOBALS ACCESSED:

toggle

USER ROUTINES CALLED:

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

grafstr

FILE:

grf.c

DESCRIPTION:

Draws a character string at current location. Leaves the current location at the end of the string. Draws the string from the lower left corner of the first letter.

PARAMETERS:

none

GLOBALS ACCESSED:

none

USER ROUTINES CALLED:

none

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

gridupdate

FILE:

GDrender.c

DESCRIPTION:

SPAWNED PROCESS that handles the updates to the GD STI
format data extraction systems.

PARAMETERS:

none

GLOBALS ACCESSED:

none

USER ROUTINES CALLED:

Setup_dma_grid

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**   hide
**
** DESCRIPTION:
**   Performs hidden point/line removal by drawing the terrain with
**   black polygons. The vw coordinate of the post type was originally
**   intended as a buffer to more efficiently align the data in memory,
**   since it is already in memory, use it to cheat the zbuffer by
**   setting it to less than vz coordinate (the actual elevation).
**   When the black polygons are drawn, use the vw coordinate for
**   elevation instead of vz.
**
** PARAMETERS:
**   None
**
** GLOBALS ACCESSED:
**   skip      i      determines the resolution of the data samples
**   tile      i      for the elevation data, and array bounds
**   xb        i      beginning x extent of view volume
**   xe        i      ending x extent of view volume
**   yb        i      beginning y extent of view volume
**   ye        i      ending y extent of view volume
**
** USER ROUTINES CALLED:
**   colour
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
**   0.1      dc robohm   92/10  added code to always draw the black
**                                squares with a valley. due to the
**                                nature of the original method, sometimes
**                                there would be peaks in the squares, and
**                                sometimes there would be valleys.
**                                the peaks would obscure anything drawn
**                                between the posts, so a method was needed
**                                to draw only valleys.
**/

```

ROUTINE:

hypot3

FILE:

math.c

DESCRIPTION:

Fast 3-D hypotenuse routine.

PARAMETERS:

a, b, c

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

Returns $\sqrt{a*a + b*b + c*c}$.

ROUTINE:

inner_prod

FILE:

math.c

DESCRIPTION:

Returns the inner product of 2 vectors.

PARAMETERS:

vector1, vector2

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

Although similar to dot, the GD version uses type double instead of float.

```

/*
** ROUTINE:
**   initgraphics
**
** DESCRIPTION:
**   Initializes the graphics window in preparation for displaying the
**   different STI formats. Also queues up the different devices that
**   will be used by the control() routine. Some of the parameters to
**   the graphics calls are hardwired to the vgx's #defines, but should
**   be set according to getgdesc() calls.
**
** PARAMETERS:
**   name      i      name of window to be opened
**   mode      i      if mode == NTSC, sets up the graphics for
**                   an ntsc signal, any other value is the full
**                   screen of the standard 60 HZ monitor
**
** GLOBALS ACCESSED:
**   aspect    i/o    aspect ratio of screen x to screen y
**   fov       i/o    field of view in screen y direction
**
** USER ROUTINES CALLED:
**   initlight
**   initmap
**
** GD ROUTINES CALLED:
**   make_chars
**
** REVISION HISTORY
**   0.0      dc robohm   92/09   original
*/

```

```

/*
** ROUTINE:
**   initlight
**
** DESCRIPTION:
**   Sets up the lighting model for doing light shaded polygons. The
**   model, light, and material are defined and bound.
**
** PARAMETERS:
**   None
**
** GLOBALS ACCESSED:
**   mat1      i      material used for lighting calculations
**   lit1      i      light used for lighting calculations
**   mod1      i      model used for lighting calculations
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09   original
*/

```



```

/*
** ROUTINE:
**   initmap
**
** DESCRIPTION:
**   Initializes an elevation based color map of the format needed to
**   pass to the cpack call (i.e., 0xAABBGRR). Not space efficient,
**   because it sets a long for each integral elevation from -100 feet
**   to 4400 feet, but it eliminates the need to do some calculations
**   at run time. Try to vary the colors to provide some texturing.
**
** PARAMETERS:
**   None
**
** GLOBALS ACCESSED:
**   sti_colrmap   o      long colormap of format 0xBBGGRR
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0          dc robohm   92/09  original
*/

```

```

/*
** ROUTINE:
**   initmat
**
** DESCRIPTION:
**   Takes a 4 X 4 double array and initializes it to an identity matrix.
**
** PARAMETERS:
**   mat[4][4] o      matrix to be initialized
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
*/

```

```

/*
** ROUTINE:
**   loadblock
**
** DESCRIPTION:
**   Loads a block of data from the file pointed to by fp into the array
**   block. If either the row or the column does not lie within the
**   terrain file, the array is filled with zeros. Routine is based
**   on GD's Load_Terrain_Block. Remember, the blocks are stored in
**   column major order within the file, and the data is stored column
**   major within the block.
**
** PARAMETERS:
**   fp          i      file pointer to opened terrain file in GD format
**   row         i      desired block row in file pointed to by fp
**   col         i      desired block col in file pointed to by fp
**   block       o      filled with elevation posts from the block
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0         dc robohm   92/09  original
*/

```

```

/*
** ROUTINE:
**   loadtile
**
** DESCRIPTION:
**   Loads the appropriate tile, block by block, into memory, sets the
**   appropriate fields of the structure, and computes the surface
**   normals of the terrain at each elevation post.
**
**   Loadtile was written to be spawned as a separate process with the
**   sproc call. There are no formal parameters, but it communicates
**   with the parent process through several fields of the array tiles.
**
**   A "double buffered" approach has been adopted to avoid the problems
**   of multiple processes concurrently accessing and modifying the same
**   memory locations. The parent process will use one buffer while
**   loadtile will modify the other buffer until processing is completed,
**   at which time the buffers are "swapped".
**
**   Loadtile will wait until the mpflag of the current buffer is set to
**   MP_CPUSTART. Once it is told to start, it will get the center row
**   and column in blocks from the blkrow and blkcol fields. Then it
**   reads data and computes normals. When all processing is done, it
**   sets mpflag to MP_CPUDONE, signalling the parent process that the
**   tile is loaded and ready for use. It then "swaps" buffers to wait
**   for another MP_CPUSTART.
**
** PARAMETERS:
**   None
**
** GLOBALS ACCESSED:
**   tiles      i/o    accesses mpflag, blkrow, and blkcol fields of
**                   the current buffer for next tile to be loaded,
**                   and loads the elevation data into the elev field.
**
** USER ROUTINES CALLED:
**   cross
**   loadblock
**   normalize
**   pp2v
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
*/

```

```

/*
** ROUTINE:
**   main (STI)
**
** DESCRIPTION:
**   The executive of the STI program. opens the DTED file, opens the
**   ethernet connection, launches the tile loader as a separate process,
**   initializes the graphics, and passes control to the controller.
**   On return from the controller, resets the graphics, kills the
**   loadtile process, closes the ethernet connection, and closes the
**   DTED file.
**
** PARAMETERS:
**   filename   i       name of the blocked DTED file. (GD's format)
**   -ntsc     i       optional. sets up the run in ntsc mode.
**   -small    i       optional. displays in an ntsc sized window
**   -ethernet i       optional. enables ethernet
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   closedmafile
**   control
**   initgraphics
**   gridupdate      (spawned as a separate process)
**   loadtile        (spawned as a separate process)
**   opendmafile
**   resetgraphics
**
** GD ROUTINES CALLED:
**   udpbclose
**   udpbopen
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
**   0.1      dc robohm   92/09  added ethernet initialization.
*/

```

ROUTINE:

make_chars

FILE:

grf.c

DESCRIPTION:

Creates the graphical text font used by the HMD as objects.

PARAMETERS:

None

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**   managememory
**
** DESCRIPTION:
**   Does the memory management job for the STI program. As input it
**   takes the current position in feet, converts it into file blocks
**   (according to GD's format), and determines if a new tile must be
**   loaded into memory. If a new tile needs to be loaded, it calls
**   loadtile indirectly through the mpflag, blkrow, and blkcol fields
**   of the current buffer of the tiles array.
**
**   This routine always looks at the same buffer of the tiles array
**   as loadtile, so it can tell loadtile to start reading in a new
**   tile, and also to catch the signal from loadtile that it has
**   completed the load. On the first invocation, managememory must
**   call loadtile and wait until loadtile has finished so the global
**   pointer tile can point to some real data. After the first call,
**   we never wait for loadtile. We simply check to see if it has
**   finished with the current load, and if it has do two things:
**   first make sure tile points to the current buffer (which contains
**   the most recently loaded data), and second check to see if we need
**   to start loading another tile. If another tile is needed, "swap"
**   buffers and give loadtile the appropriate parameters in the tiles
**   array. Let loadtile do its thing while managememory returns
**   control to the calling routine. If loadtile wasn't finished with
**   its current tile, simply return control to the calling routine.
**
** PARAMETERS:
**   x      i      east - west position in feet
**   y      i      north - south position in feet
**
** GLOBALS ACCESSED:
**   tile    o      pointer to the current tiles buffer.
**   tiles   i/o    mpflag, blkrow, and blkcol fields provide
**                   communication with the loadtile process.
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09    original
*/

```

```

/*
** ROUTINE:
**     mpymat
**
** DESCRIPTION:
**     Performs matrix multiplication  $c = a * b$ . originally set a
**     temporary matrix to  $a * b$ , then sets  $c = tmp$ , in case c is
**     the same matrix as a or b.
**
** PARAMETERS:
**     a[4][4]      i      a matrix
**     b[4][4]      i      b matrix
**     c[4][4]      o      result of  $a * b$ 
**
** GLOBALS ACCESSED:
**     None
**
** USER ROUTINES CALLED:
**     None
**
** REVISION HISTORY
**     0.0          dc robohm    92/09  original
*/

```



```
/*
** ROUTINE:
**   normalize
**
** DESCRIPTION:
**   Normalize a vector to unit length. Replaces the original vector
**   with the unitized one.
**
** PARAMETERS:
**   v      i/o    vector to be normalized
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
**/
```

ROUTINE:

normalize_vec

FILE:

math.c

DESCRIPTION:

Normalizes a vector to unit length.

PARAMETERS:

vector

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

hypot3

REVISION HISTORY:

0.0 GD original.

NOTES:

Although similar to normalize, the GD version uses type double instead of float.

```

/*
** ROUTINE:
**   opendmafile
**
** DESCRIPTION:
**   Opens the given dma file for reading, and sets dmafp to point to
**   the file pointer.
**
** PARAMETERS:
**   name      i      name of DTED file to open
**
** GLOBALS ACCESSED:
**   dmafp     o      file pointer to the dted file
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm   92/09  original
*/

```

ROUTINE:

outer_prod

FILE:

math.c

DESCRIPTION:

Computes the outer product of 2 vectors.

PARAMETERS:

vector1, vector2, result

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

Although similar to cross, the GD version uses
type double instead of float.

```

/*
** ROUTINE:
**   pp2v
**
** DESCRIPTION:
**   Determines the vector between two points.
**
** PARAMETERS:
**   p1      i      point 1
**   p2      i      point 2
**   v      o      vector from point 1 to point 2
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
*/

```

ROUTINE:

Radar_alt

FILE:

ethernet.c

DESCRIPTION:

Radar_alt determines 1) if the radar altimeter CAN
determine the radar altitude and, if so,
2) what the radar altitude is.

PARAMETERS:

os

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

ROUTINE:

rcv_ethernet

FILE:

ethernet.c

DESCRIPTION:

Rcv_ethernet picks up and decodes packets sent to the open socket. Only packets with the correct id code, i.e. one == thisship will be recognized. There are 3 data packets that are used by the STI system: TERRAIN type, which include such things as the terrain file to load, the x coordinate of the western edge of the database, the y coordinate of the southern edge of the database, and the STI parameters, and 2) the TARGET type, which reads a packet of etherown type into a structure of target_type. (The first portion of target_type is identical to etherown_type). Angle_computations then expands the data to fill the rest of the target structure.

PARAMETERS:

None

GLOBALS ACCESSED:

os terrparm gc_steering wind planetime netfd thisship terrain_name

USER ROUTINES CALLED:

getelev

Radar_alt

Angle_computations

REVISION HISTORY:

0.0 GD original.

NOTES:

```

/*
** ROUTINE:
**   render
**
** DESCRIPTION:
**   Takes as input, the desired format to be drawn, and calls the
**   appropriate routines to display that format. The symbolic
**   constants for the formats are defined in def.h.
**
** PARAMETERS:
**   format      i      the format of the terrain display
**
** GLOBALS ACCESSED:
**   fat         i      flag to determine if lines are 1 or 2 pixels
**   hidden      i      flag to determine if we need to do hidden
**                       point/line removal
**   position    i      to get height AGL for progressive format
**
** USER ROUTINES CALLED:
**   Draw_dma_grid
**   Draw_dma_ridge
**   drawgradient
**   drawmesh
**   drawpoints
**   drawpolys
**   drawsquares
**   hide
**
** REVISION HISTORY
**   0.0         dc robohm   92/09  original
*/

```



```

/*
** ROUTINE:
**   resetgraphics
**
** DESCRIPTION:
**   Resets the graphics to a reasonable state when the simulation is
**   completed.
**
** PARAMETERS:
**   mode      i      if mode == NTSC, an ntsc signal was being
**                   output, so it must be reset to the standard
**                   60 HZ monitor
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   None
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
*/

```

ROUTINE:

Ridgeline_extract

FILE:

GDrender.c

DESCRIPTION:

Ridgeline extract is very similar to Get_high_intersections, but instead of storing the point representing the highest visual angle to date based on crossing some criteria, Ridgeline extract stores the point whenever a ridgeline is crossed, that is, whenever the visual angle to the projected point is less than the angle to the previous point. Logic then tries to correlate the point with a ridgepoint from the previous projection; if one is found, the point is added to its array; else a new array is started. If, after the projection reaches "far" there is an array of points which was NOT added to, it is "closed" and the remainder of the array made available for re-use. These points are stored in the same h and vl arrays used by Get_high_intersections, allowing up to 8 separate ridges along one angle.

PARAMETERS:

point --the starting point of the projection.
los --unit vector defining line of sight to look down.
far --for 4th horizontal line
vdf --proportion of the projected line in the
--direction of the LOS.
h --array holding the horizontal line points.
v --array holding the vertical line points.
nph --number of points in the horizontal line arrays.
npv --number of points in the vertical line arrays.
last_hratio --last visual angle from previous projection.
last_vratio --same for the vl array.

GLOBALS ACCESSED:

toggle

USER ROUTINES CALLED:

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

rotate_vector

FILE:

math.c

DESCRIPTION:

Rotates a vector using passed matrix.

PARAMETERS:

vector, rot_mat

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

See set_rotation above.

```

/*
** ROUTINE:
**   rotmat
**
** DESCRIPTION:
**   Rotates the matrix by a radians around the given axis. This routine
**   multiplies the matrices in the same order as SGI's rot() call.
**
** PARAMETERS:
**   a      i      angle (in radians) to rotate the matrix by
**   axis    i      axis about which the matrix is to be rotated
**   mat[4][4] o    matrix to be initialized
**
** GLOBALS ACCESSED:
**   None
**
** USER ROUTINES CALLED:
**   initmat
**   mpymat
**
** REVISION HISTORY
**   0.0      dc robohm    92/09  original
*/

```

ROUTINE:

rotvec2d

FILE:

GDrender.c

DESCRIPTION:

Rotvec2d is a 2 dimensional version of rotate_vector.
Since it assumes rotation around "z", and only affects the
xy of the vector, it can be optimized for 2d use.

PARAMETERS:

ang, vec

GLOBALS ACCESSED:

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

set_rotation

FILE:

math.c

DESCRIPTION:

Sets up the matrix for vector rotation using rotate_vector.

PARAMETERS:

u, angle, rot_mat

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

Sets up the matrix to rotate around unit vector u by angle radians and returns it in rot_mat. Better than rotating directly, since frequently several vectors will need to be rotated identically.

ROUTINE:

Setup_dma_grid

FILE:

GDrender.c

DESCRIPTION:

Depending on whether the STI format is ortho or a ridgeline extraction system, calls Get_high_intersections or Ridgeline_extract for angles off the current line of sight, extracting the information needed to draw the lines representing the format and storing them in the h, vr, and vl arrays. When the array is through updating, flips toggle to allow access to the new data.

PARAMETERS:

os eye_ul newar far format

GLOBALS ACCESSED:

h, vl, vr, npv, npvr, npvl, toggle

USER ROUTINES CALLED:

Get_high_intersections

Ridgeline_extract

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

udpbread

FILE:

udpb.c

DESCRIPTION:

Reads a packet from the open socket.

PARAMETERS:

netfd buffer len

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

ROUTINE:

udpbopen

FILE:

udpb.c

DESCRIPTION:

Opens the socket for the UDP protocol ethernet system. The socket opened must be defined in the /etc/services file of the system.

PARAMETERS:

service (string: name of the service).

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

Getbroadcast

Gethostaddr

REVISION HISTORY:

0.0 GD original.

NOTES:

Getbroadcast and Gethostaddr were modified from the code provided by Silicon Graphics in the /usr/people/4Dgifts/examples/network directory.

ROUTINE:
 udpclose
FILE:
 udpb.c
DESCRIPTION:
 Closes the open socket.
PARAMETERS:
 netfd
GLOBALS ACCESSED:
 None
USER ROUTINES CALLED:
 None
REVISION HISTORY:
 0.0 GD original.
NOTES:

ROUTINE:

udpwrite

FILE:

udpb.c

DESCRIPTION:

Writes a packet to the open socket.

PARAMETERS:

netfd buffer len

GLOBALS ACCESSED:

None

USER ROUTINES CALLED:

None

REVISION HISTORY:

0.0 GD original.

NOTES:

APPENDIX B

Concept Paper

INTRODUCTION

The objective of this effort is to evaluate the utility of a synthetically derived terrain image on a pilot's helmet-mounted display (HMD) from stored digital terrain data for the purpose of improved Situation Awareness.

In order to accomplish the stated objective, it will be necessary to consider the technology issues associated with the display of synthetic terrain imagery (STI) formats. In addition, considerable effort is required (design and evaluation) to assure that the pilot can easily interpret the terrain formats while flying high speed tactical missions.

The purpose of this Concept Paper is to document the rationale for the approach taken by the Honeywell/General Dynamics team in the pursuit of program goals.

The remainder of this brief is organized as follows:

Mission Requirements:	Outlines the mission needs that impact the terrain format design
User Needs:	Review STI issues associated with spatial situation awareness
Format Concepts:	Presentation of the STI formats developed for testing
PVI Analysis:	Discussion of the integration of STI formats with conventional alphanumeric symbology
Recommended Approach:	Outlines the STI formats that will be taken into testing

MISSION REQUIREMENTS

The capability to employ day tactics on night missions has always been recognized as extremely beneficial, but only recently has the technology evolved to the point where this is now becoming feasible.

The operational requirements for the night close air support/battlefield air interdiction (CAS/BAI) mission remain the same as for day missions. For example, the aircraft must penetrate the threat environment and navigate to the target area. Once there, the pilot must acquire the target, properly maneuver the aircraft to deliver selected weapons, and safely egress.

An effective night attack system would allow the pilot to ingress the target area at low altitude and then use a delivery mode such as Continuously Computed Impact Point (CCIP) for weapons employment. This delivery mode, which requires the pilot to visually acquire the target, is difficult enough during day operations but nearly impossible at night without an effective night attack system. Egress from the target area will again be at low altitude to minimize detection by enemy threats.

An effective night attack system should allow the pilot to utilize tactics similar to those used during the day. For example, a conventional daytime CAS scenario begins with a low-level ingress to a contact point to receive target information from a Forward Air Controller (FAC). This target information consists of a "9-line briefing" which includes:

- location of IP
- heading and distance to target
- target location

- target elevation
- target description
- how the target will be identified (smoke, laser, etc.)
- location of friendly troops
- threat location
- egress direction

The attack aircraft proceeds to the IP to attack the assigned target. Ingress to the target area will be at low altitude taking advantage of terrain features. Radio emissions are minimized to mask the intended approach to the target.

Since ingress altitude is usually low to avoid detection, a pop-up maneuver is required to aid in target acquisition. At four to five nautical miles from the target, a 3 to 4G turn is executed to offset the target 30-40 degrees left or right of the nose. An offset attack is used to allow the pilot more time for target acquisition and better control of the apex altitude during the pop, based on target/aircraft attack geometry. Weapon release is accomplished in a low angle dive delivery followed immediately by a hard turn to avoid the weapon fragmentation pattern. Chaff/flare expendables and hard maneuvering are also required after weapon release to defeat enemy defenses.

To provide mutual support and increase the number of bombs on target, a formation of two aircraft will normally be employed. When two aircraft are used to attack a target, each member of the formation must have knowledge of the other aircraft's position during the low altitude ingress, target attack, and egress. The wingman's position during the ingress can be from line abreast to as much as 60 degrees aft of the leader based on the threat and surrounding terrain. When the leader offsets, the wingman will offset to gain separation in both attack heading and time from the leader. This will normally require the wingman to delay his pull-up or arc the target at 3-4 nautical miles prior to beginning his pop-up maneuver. This separation varies the attack axis and helps the wingman avoid the fragmentation of the leader's weapons. The egress maneuvers are similar with the intent of providing mutual support as quickly as possible coming off the target.

For guided munitions such as Maverick (AGM-65) missiles, the attack profile would be similar to those previously described except the range for offset/arcing of the target area would be greater and the altitude in the climb would be the minimum required to acquire the target. Since the field-of-view of guided munitions is very narrow, it is critical that the target be present in the weapons field-of-view for manual designation, or automatically locked-on when line-of-sight is established for a successful first pass attack. To minimize closure on the target, and hence maximize stand-off range, an offset attack near the weapons azimuth lock-on limits should be employed. This will allow multiple weapons to be employed at targets in close proximity, minimize the total time that the aircraft is exposed to surface-to-air threats, and keep the aircraft at the greatest standoff distance from the target.

To successfully accomplish the above mission at night, the pilot must be able to acquire the intended target in sufficient time to complete the attack profile. For employing conventional weapons, a night vision system must provide sufficient resolution to have three miles visibility in most conditions and provide sufficient warning of obstacles and cultural features. During the attack phase, the system must have sufficient resolution to locate a target at ranges of 4 to 5 nautical miles and provide target recognition prior to weapon release. Positive target acquisition in narrow field-of-view must occur between 2 to 3 nautical miles to allow 8 to 10 seconds to position the aircraft for weapons delivery.

Weapons release will occur at ranges slightly less than 1 nautical mile from dive angles of 5 to 10 degrees.

Azimuth requirements for utilizing a Head-Steered Infra Red (HSIR) sensor will exceed 60 degrees during several segments of the attack profile. For example, during target ingress, the wingman could require azimuth angles exceeding 90-degrees to determine the leader's position. The wingman's azimuth to the target may exceed 60-degrees while maneuvering to retain the appropriate interval and spacing from the leader by delaying the pull-up in the target area. The leader may also require more than 90-degree azimuth coverage to acquire the wingman's position during the egress for mutual support and rejoin.

In addition to an effective night vision system, there are two other key elements needed to ease pilot workload during night operations—accurate navigation and positive target acquisition. Accurate navigation can be provided by incorporating a Digital Terrain System (DTS) which will reduce the time spent on navigation. The DTS can also provide covert terrain-following (TF) capability, an all attitude predictive ground proximity warning system, and passive ranging. This system will also form the basis for a Synthetic Terrain Imaging system.

An effective data link system can ease the target acquisition problem. This system would use the UHF or VHF radio to receive data link information from either a ground observer or similarly equipped fixed wing/rotary aircraft. Using a data link transmission, the FAC passes the "9-line brief" directly to the Fire Control Computer and graphically displays it to the F-16 pilot. The pilot now has IP/target location, desired run-in heading, target type, and location of friendlies displayed without the need for lengthy voice communications and subsequent manual entry of targeting data.

USER NEEDS

Goals for the terrain images are: formats sufficient for terrain awareness during high speed low altitude flight, day or night, when the horizon is indistinct or obscured, and height above the surface cannot be judged visually. Display formats should provide terrain awareness over any surface. The formats should project minimal imagery necessary to provide the pilot with sufficient visual cues to determine approximate height above the surface, approximate distance to points on the surface, the motion of the aircraft relative to the surface, and, the relative proximity, shape, and size of terrain features. The terrain image needs to be detailed only enough to provide awareness of general terrain features. The pilot needs to perceive the terrain, but not have the image block or interfere with his view of the outside world or the cockpit. The image should be available in the full field of view of the HMD, and must be available not only in the direction of flight, but off-axis enough to permit aggressive, high-G maneuvering for terrain masking during the navigation phases of flight, as well as during offensive and defensive maneuvering. Enhancements of the raw terrain image with other features to add to the pilot's total situation awareness should be explored. The design effort must involve the consideration of pilot visual perception, and other human factors.

The intent of the STI for HMD program is to provide the pilot with earth references to reduce the possibility of disorientation, particularly with respect to sky/ground orientation. In order to achieve the desired program goal, "synthetically" derived terrain depiction's will be rendered on the HMD to show the relative position of the earth (spatial relationship to aircraft and proximity) and gross features of the terrain.

It is useful to examine how STI formats can aid in the fulfillment of mission requirements. The following Table identifies the elements of the STI system (including Off-Axis Attitude Awareness Symbolology) that aid the pilot in accomplishing the mission requirements.

	Off-Axis Attitude Awareness (OAAS) Symbolology	Terrain Depiction	Horizon (Up/Down) Orientation	Motion Cues
Low Altitude Flight	X	X	X	X
Navigation	X			
Terrain Masking		X		X
Weapon Delivery	X	X		X
Pop-up Maneuver	X	X	X	X
Target Acquisition	X			

STI aids to Mission Requirements

It has long been known that without reference to outside cues pilots will quickly become disoriented. Jarvi (1981), in a report investigating spatial disorientation in F-15 pilots, compiled a list of factors that contribute to disorientation, including:








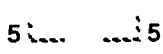

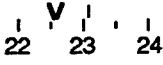

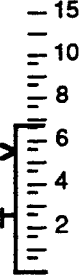


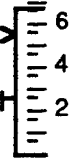








- Autokinesis
- Fascination
- Target Hypnosis
- Illusory Effects Due to Inadequate Stimuli
- Improper Grouping of Lights at Night
- Illusions of Relative Motion
- Illusory Horizons

It is the intent of the current program to overcome visual factors that contribute to spatial disorientation by rendering a synthetic image of terrain (or water) so the pilot has a clear, unambiguous, indication of orientation with respect to the earth. A reasonable concern with the current project is the addition of symbolology to a display which already has a number of discrete elements for fear overwhelming the pilot. To state this more directly, the pilot may not have the attentional resources required to interpret the STI. Recently Weinstein and Wickens (1992) have shown that an "ecologically" valid display (contact analog of the out-the-window visual scene) allowed for the most efficient time-sharing.

FORMAT CONCEPTS

Helmet-Mounted Display Symbolology

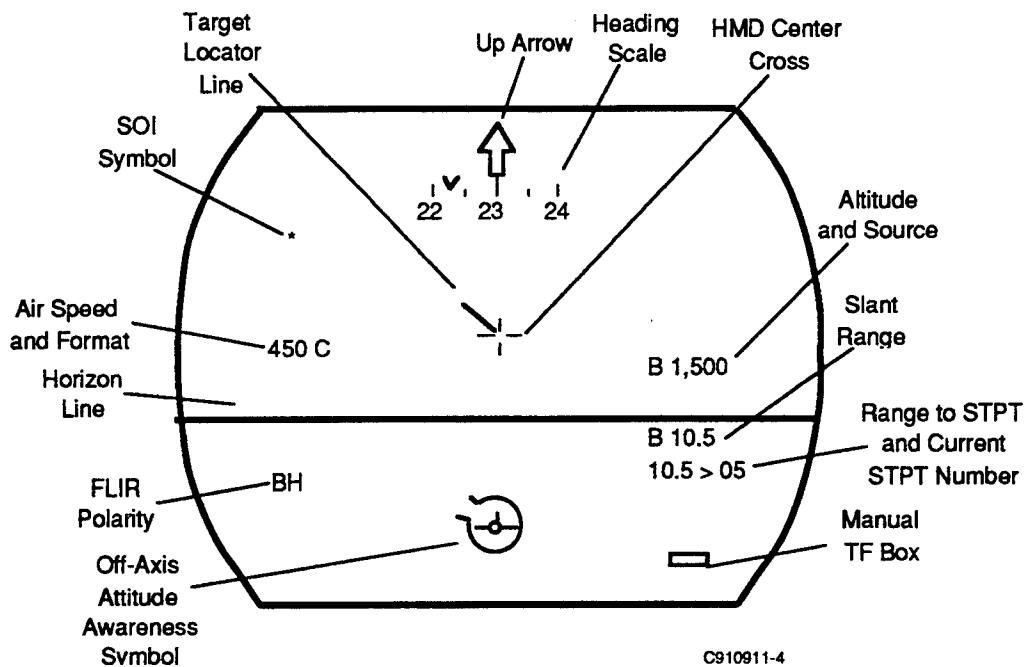
The STI formats will be overlaid with the HMD symbol set that was developed by General Dynamics during the "Night Attack Program Pilot-Vehicle Interface Study." The HMD symbol set is shown below:

Original HMD Symbolology Set		Expanded HMD Symbol Set	
Boresight Cross		TF Cue	
Centercross		Angle of Attack (ADA) Error Bracket	
Horizon Line			
Up Pointer		Pitch Scale	
A/G Target Designator		Heading Scale	
Target Locator Line			
CCIP Reticule			
Bomb Fall Line with Solution Cues		Radar Thermometer Scale	
Breakaway Cross			
Flight Path Marker			
Pull-up Anticipation Cue			
FLIR Polarity	BH		
Airspeed	350 C		
Altitude	R 1,000		
Warning	WARN		
Fuel	FUEL		
FLIR Overheat	OVHT		
LOW error	LAG		
FLIR Gain	G 03		
FLIR Level	L 45		
NFOV Corners*		TERPROM Status Window	

* NFOV corners are generated by the FLIR and are embedded in the raster picture

C910911-09

An example of the "Night Attack" symbology in the HMD is shown below.



Synthetic Terrain Imagery Formats

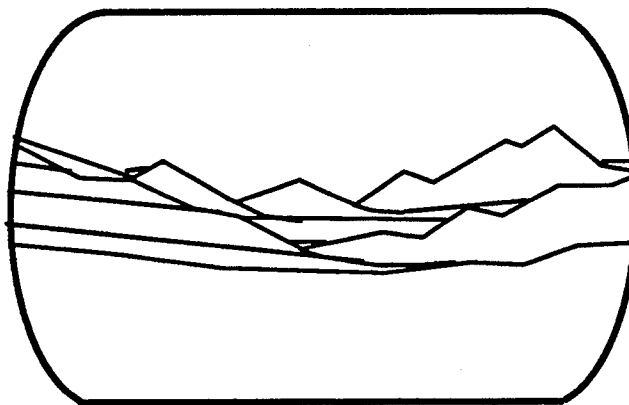
It is important to identify the elements of the terrain image that aid in the determination of spatial orientation to assure that the STI formats contain these elements. Weinstein and Wickens (1992) have provided a concise review of the format characteristics of a contact analog display:

- Optical Flow:** The accelerating flow of texture down the visual field
- Splay:** Parallel lines with a common vanishing point
- Compression:** The foreshortening of a projection as it is slanted away from the frontal plane

A key element in the presentation of the STI presentation is the frame of reference for the rendering of the terrain image, it can either be geographically fixed or aircraft centered. If the format is observer centered then the presentation will always have the cockpit as the center of orientation (a bit like the early, Ptolemy, view of the Universe). The geographically fixed presentation requires that the format elements (e.g., lines, points of light, tiles) be linked to the ground.

The first two formats, Ridgelines and the Optical Expansion Gradient, are aircraft centered formats. The remaining formats are geographically fixed.

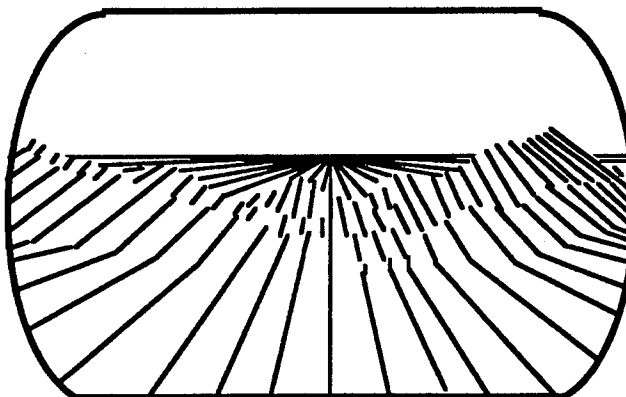
Ridgelines —



C82057-17

This is format an outgrowth of the work Honeywell conducted on the Quiet Knight program. Discrete range-bins from the aircraft are sampled. Within each range-bin the highest points are identified and a line is drawn laterally connecting the peaks.

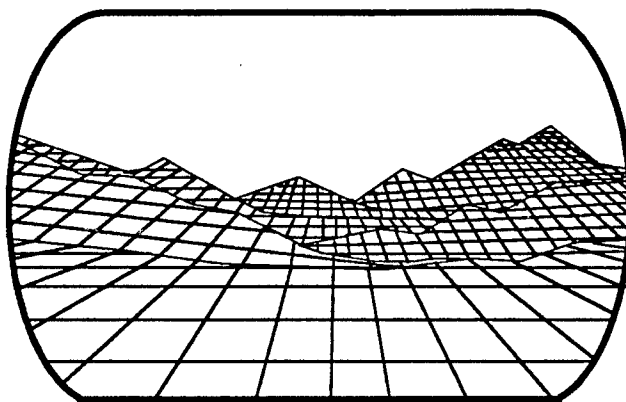
Optical Expansion Gradient —



C820574-01

This approach is an analog of the splay lines in an attitude direction indicator (ADI). The implementation of this approach, similar to Ridgelines, requires an observer centered (as opposed to a geographically fixed) approach. This is because the vanishing point must always be centered on the flight path vector of the aircraft. The highest points, at a fixed spacing, are then represented by perturbations in the splay lines of the gradient.

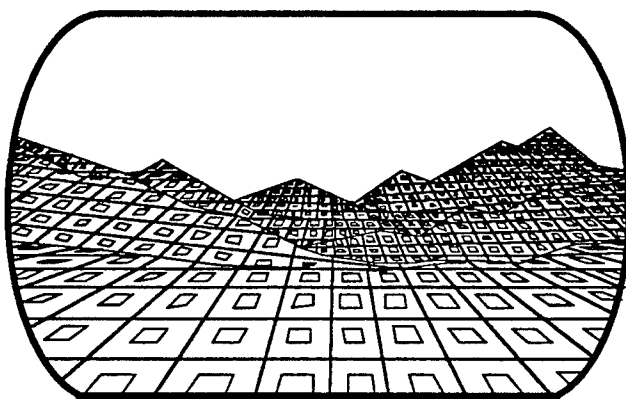
Contact Analog Grid —



C920574-03

The geographically fixed presentation scheme looks as though a large net were laid across the earth. The grid is deformed by topographic surface features.

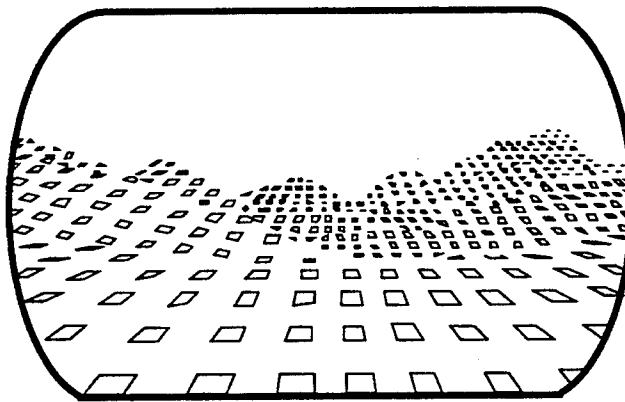
Contact Analog Grid with Tiles —



C920574-03

This presentation scheme is identical to the Contact Analog Grid with the addition of Tiles in the open area within each grid. The tiles are oriented perpendicular to the surface normal of the local square.

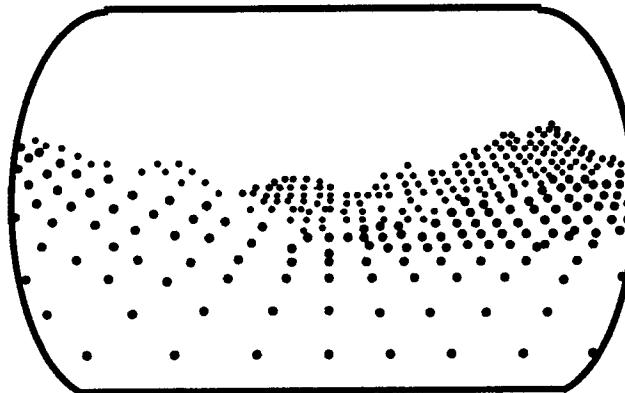
Tiles —



C820574-04

This presentation scheme is identical to the Contact Analog Grid with Tiles except the grid is missing.

Posttops —



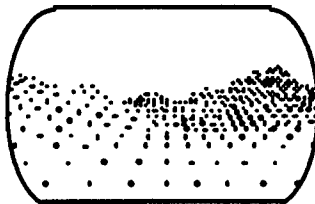
C820574-02

The Posttops format consists of lighting each sample point in the DTED database at the appropriate elevation in the perspective view volume.

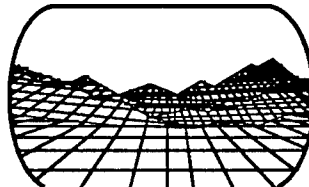
Emergent Detail —

An approach to cueing the pilot to critical altitude changes (i.e., penetration of the "harddeck") is change formats. The use of emergent detail is intended to cue the pilot to a change in situation without requiring a great deal of cognitive processing.

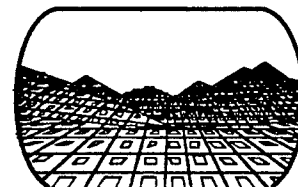
The use of emergent detail to cue terrain proximity is not new (Reardon and Warren, 1989), but the application to an HMD would be unique. An example strategy for the use of emergent detail would be to present Posttops when above 2,000 feet AGL. The Posttops would change to the Contact Analog Grid below 2,000 feet, and Tiles would be added to the Grid below 500 feet AGL.



Above 2,000 ft AGL



Below 2,000 ft AGL



Below 500 ft AGL

PILOT-VEHICLE INTEGRATION

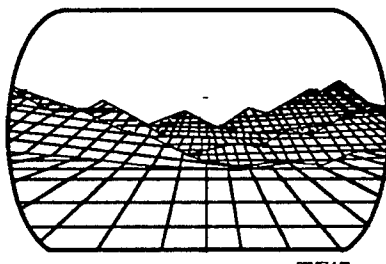
Acceptance of the display, in terms of subjective pilot opinion, is pivotal for implementation of STI in helmet-mounted displays in an operational environment. Display factors that will affect pilot acceptance of the display include Depth Cueing, Sampling Density, View Volume, and Anti-Aliasing. All of these factors will have an impact on processing requirements and graphics throughput which in turn impact real-time performance of the display.

Depth Cueing

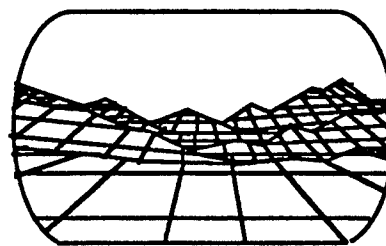
By varying the intensity of STI elements (distant portions of the terrain are shown dimmer) the real-world phenomenon of blurring of distant objects can be achieved. It is also a means by which terrain entering the display view volume can "gracefully" come into view, as opposed to the sudden appearance of a new STI elements. Depth cueing requires that a light source and appropriate shading algorithms are executed.

Sampling Density

The data base supplying terrain information, U.S. Defense Mapping Agency's Digital Terrain Elevation Data (DTED), provides terrain elevation in 100 meter increments. Therefore the greatest resolution available to STI is 100 meters. Sampling density has a direct effect on speed of graphics processing, the higher the density sampled the slower the graphics processing (more data slows the processing).



CHIEF 4-02



CHIEF 4-01

View Volume

The "distance" from the closest terrain depiction to the STI horizon is called the view volume of the display. View volume can be selected, ranging from 1 to 6 nautical miles. View volume has a direct effect on speed of graphics processing, the greater the view volume the slower the graphics processing (more data slows the processing).

Anti-Aliasing

There are a number of subjective factors that contribute to the perception of display quality. A key element is the "smooth" rotation transition of a line from horizontal to vertical. Aliasing occurs when a line becomes "jagged" during this rotation. There are different methods by which anti-aliasing can be accomplished, including adding separate, dedicated, processing hardware.

Real-Time Operation (Update Rate)

During preliminary format development it became clear that maintaining an update rate of at least 15hz is difficult when the STI format was rendered from the highest sample density (100 meter spacing) with a large view volume (6 miles). In the current implementation sample density can be set to one of the following settings (100, 200, 400, or 800 meter). The view volume can vary, in tenth of a mile increments between 1 and 6 miles.

One of the key elements in the initial tests in the PVI development station at General Dynamics will be to determine the lowest acceptable update rate for the display.

RECOMMENDED APPROACH

The Honeywell/General Dynamics team will conduct evaluations of the Synthetic Terrain Imagery formats in both part-task and full mission simulation. Operationally oriented mission scenarios will be flown under varied terrain and visibility conditions to determine if STI aids the pilot with spatial orientation. Mechanization's for controlling STI viewability/intensity will also be investigated.

PART-TASK SIMULATION

In an attempt to eliminate unnecessary test conditions for the actual evaluation, General Dynamics will utilize in-house test pilots and former F-16 pilots to determine which STI techniques have high potential. The following Table is the initial "cut" at the relative merits and disadvantages to each approach.

	Pro	Con
STI FORMATS		
Ridgelines	Previous flight test experience Quiet Knight	Aircraft centered "SQUIRM"
Optical Expansion Gradient	Looks like the OEG on an analog Attitude Direction Indicator (ADI)	From altitude, with -90°, the splay lines provide minimal info
Contact Analog Grid	Intuitive	
Grid & Tiles	Terrain features stand out Can be a "virtual" VR world	May be too cluttered for un-aided target acquisition
Tiles	Tile orientation alone may be adequate for terrain SA	Obscures terrain information with low sampling density
Posttops	Visually least intrusive	Terrain features hard to discern

Relative Merits of STI Formats

For the actual evaluation, the F-16 SPO, as well as selected active duty F-16 pilots will be exposed to the high potential STI formats in a part-task simulation evaluation.

FULL-MISSION SIMULATION

The best techniques will be transferred to the dome where they can be evaluated utilizing combat mission profiles. Each participating pilot will evaluate the STI techniques. Each mission will be verbally debriefed and a questionnaire completed. Additionally, objective mission data will be collected and compared to the subjective data obtained from the questionnaires. Conclusions and recommendations will then be derived based on the objective and subjective data collected.

Selected Readings

- Biberman, L. M. and Alluisi, E. A. (1992). Pilot errors involving head-up displays (HUDs), helmet-mounted displays (HMDs), and night vision goggles (NVGs). *IDA Paper P-2638*. Institute for Defense Analyses, Alexandria, VA.
- Haber, R. N. (1987). Why low-flying fighter planes crash: Perceptual and attentional factors in collisions with the ground. *Human Factors*, 29(5), 519-532.
- Hale, S. and Piccione, D. (1989). Pilot assessment of the AH-64 helmet mounted display system. *Proceedings of the Fifth International Aviation Psychology Symposium*, pages 307-312.
- Ikeda, M. and Takeuchi, T. (1975). Influence of foveal load on the functional visual field. *Perception & Psychophysics*, 18(4), 255-260.
- Jarvi, D. W. (1981). Investigation of spatial disorientation of F-15 Eagle pilots. *Technical Report ASD-TR-81-5016*, Directorate of Equipment Engineering, Air Force Systems Command, Wright-Patterson AFB, OH.
- Malcolm, R. (1984). Pilot disorientation and the use of a peripheral vision display. In: *Aviation, Space, and Environmental Medicine*, Aerospace Medical Association, Washington, D.C.
- Newman, R. L. (1987). Responses to Roscoe, "The trouble with HUDs and HMD." *Human Factors Society Bulletin*, Vol 30..
- Poppen, J. R. (1936). Equilibratory functions in instrument flying. *The Journal of Aviation Medicine*, 7, Aero Medical Association of the United States.
- Reardon, K. A. and Warren, R. (1989). Effect of emergent detail on descent-rate estimations in flight simulators. *Proceedings of the Fifth International Aviation Psychology Symposium*, pages 714-719.
- Roscoe, S. N. (1987). The trouble with HUDs and HMDs. *Human Factors Society Bulletin*, Vol. 30.
- Roscoe, S. N. (1987). The trouble with virtual images revisited. *Human Factors Society Bulletin*. Vol. 30.
- Weinstein, L. F. and Wickens, C. D. (1992). Use of Nontraditional Flight Displays for the Reduction of Central Visual Overload in the Cockpit. *The International Journal of Aviation Psychology*, 2(2), 121-142.
- Weintraub, D. J. (1987). HUDs, HMDs, and common sense: Polishing virtual images. *Human Factors Society Bulletin*, Vol. 30.
- Williams, L. J. (1982). Cognitive load and the functional field of view. *Human Factors*, 24(6), 683-692.
- Williams, L. J. (1985). Tunnel vision induced by a foveal load manipulation. *Human Factors*, 27(2), 221-227.

Addendum to Appendix B

(cited from Jarvi, 1981)

VISUALLY INDUCED SPATIAL DISORIENTATION

Orientation from the external scene during flight depends upon perception of complex and continually changing patterns of visual stimuli. The validity and accuracy of both the perception and the interpretation of these cues is a function of the aviator's experience and training. Attitude is judged by reference to the horizon or when nearer the ground, by the verticals of buildings, masts, and trees. Distance and depth are determined principally by monocular cues such as parallax displacement, aerial perspective, apparent size, and by changes in both detail and color with distance (Benson, 1965).

Unfortunately, outside visual references are often reduced by smoke, haze, fog, inclement weather, or darkness. In such situations the pilot's interpretation of visual cues becomes more difficult, illusory visual information may occur, and visual phenomena themselves may contribute to disorientation. Examples of these types of spatial disorientation (adapted from Peters, 1968) are listed below:

(a) *Autokinesis*. This illusion consists of an apparent motion of isolated lights viewed in a meager visual framework. If an isolated light is viewed continually in the dark, it will appear to wander about at random over a small area. The apparent motion may extend as much as 15 degrees and is indistinguishable from real motion. Pilots have reported attempts to join up with a formation of stars, buoys, lights on bridges, and street lights which appeared to be moving and were interpreted as other aircraft.

(b) *Fascination*. This is a condition in which the pilot fails to respond adequately to a clearly defined stimulus situation in spite of the fact that all of the necessary cues are present for a proper response, and the correct procedure is well known to him.

(c) *Target Hypnosis*. Target hypnosis is a form of fascination and is characterized by a pilot becoming so intent on destroying the target during an attack that he fails to pull up in time to avoid striking the ground, usually with fatal consequences.

(d) *Illusory Effects Due to Inadequate Stimuli*. Restriction of the visual field by smoke, dust, haze, fog, rain, or darkness can produce gross discrepancies between physical entities and their appearance as perceived by the pilot. The pilot's attempt to restructure the physical entity from his meager perception of it may result in a false identification and consequent disorientation.

(e) *Improper Grouping of Lights at Night*. The tendency to group items in the perceptual field can contribute to illusory effects. A small cluster of isolated lights on the ground on a dark night with a high overcast may be interpreted as the lights of a formation flight.

(f) *Illusions of Relative Motion*. Experience of illusions of relative motion are numerous. To an observer in a fast aircraft crossing the path of a much slower aircraft at a different altitude, the slower aircraft appears to be flying sideways and backwards. Illusions of relative motion can be especially provocative and potentially hazardous during formation flights at high altitude or at night when cues to forward speed are absent.

(g) *Illusory Horizons.* The primary cue to the vertical is the visible horizon; using this cue the pilot can orient his aircraft properly and with great precision. Under conditions of restricted visibility the horizon may become obscure or occulted. Under these conditions the pilot may rely on some other indicator which he believes to represent the horizontal. Under certain other conditions and in perfectly clear weather the pilot may orient his aircraft improperly despite using the visible horizon as a reference. Various types of disorientation may be produced by reliance on fictitious horizons (e.g., tilted cloud banks; depressed horizons due to high altitude flight; confusion between city lights and stars).